



Grant Agreement 224442

Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2)

Report type	Deliverable D2.1
Report name	Appendix 3.1 Requirements and V&V Support
Dissemination level	PU
Status	Final
Version number	1.1
Date of preparation	2010-06-03

Authors**Editor**

Matthias Weber

E-mail

Matthias.Weber@carmeq.com

Authors

Andreas Abele, Conti

Lars-Olof Berntsson. VTEC

Matthias Weber, Carmeq

E-mail

Andreas.Abele@continental-corporation.com

Lars-Olof.Berntsson@volvo.com

Matthias.Weber@carmeq.com

The Consortium

Volvo Technology Corporation (S)	VW/Carmeq (D)	Centro Ricerche Fiat (I)
Continental Automotive (D)	Delphi/Mecel (S)	
Mentor Graphics Hungary (H)	CEA LIST (F)	
Kungliga Tekniska Högskolan (S)	Technische Universität Berlin (D)	University of Hull (GB)

Revision chart and history log

Version	Date	Reason
1.0	26.02.09	Release of major version based on several iterations
1.1	3.6.2010	Public Version

Table of contents

Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2)	1
Authors	2
Revision chart and history log	3
Table of contents	4
1 Introduction	6
2 Evaluation and update suggestions for requirements and V&V concepts	9
2.1 Language modifications	9
2.2 Profile related modifications	9
2.3 EAST-ADL2 support for contextual allocation and contextual ownership	9
2.3.1 <i>Elaboration around definitions</i>	10
2.3.2 <i>EAST-ADL2 contextual allocation; allocation of traceable specifications to a specific context</i>	10
2.3.3 <i>EAST-ADL2 contextual ownership; owned relationship and owned comment</i>	11
2.3.4 <i>EAST-ADL2 contextual owned satisfy relationship</i>	12
2.4 Requirement engineering with EAST-ADL2	12
2.4.1 <i>EAST-ADL2 solution-free requirements-handling</i>	13
2.4.2 <i>Formulation of EAST-ADL2 requirements</i>	13
2.4.3 <i>EAST-ADL2 use-case model</i>	13
2.4.4 <i>Stakeholders and stakeholder needs in EAST-ADL2</i>	13
2.4.5 <i>Use case relation to product features in EAST-ADL2</i>	14
2.4.6 <i>Using Requirements Interchange Format (RIF) with EAST-ADL2</i>	15
2.5 Traditional requirement process as input for update suggestions in EAST-ADL2	22
2.5.1 <i>Business opportunity, problem statement and product positioning in EAST-ADL2</i>	23
2.6 EAST-ADL2 support for requirement review	23
2.6.1 <i>Requirement review checklist in EAST-ADL2</i>	24
2.7 Relation to non-software RM and QA Models	25
2.8 Information Exchange with RM/QA tools	25
2.9 Analysis and Evaluation of EAST-ADL Support for Timing Requirements	26
2.9.1 <i>TIMMO Timing Concepts</i>	27
3 Contribution to overall ATESST2 objectives	29
4 Conclusions	30
5 Annex 1: SysML	31
5.1.1 <i>Normative References</i>	31
5.1.2 <i>Relationships to Other standards</i>	31
5.1.3 <i>SysML Language Architecture</i>	32
5.1.4 <i>SysML Language Formalism</i>	32

5.1.5	<i>Levels of Formalism</i>	32
5.1.6	<i>State of SysML Requirement and Test Case Concepts</i>	32
6	Annex 2: EAST-ADL2 Requirements	41
6.1.1	<i>EAST-ADL2 - Requirements Construct, «ADLRequirement«</i>	41
6.1.2	<i>EAST-ADL2 – Requirements Container Construct, «ADLRequirementContainer«</i>	42
6.2	Annex 3: EAST-ADL2 Infrastructure.....	42
6.2.1	<i>Package Context</i>	42
6.2.2	<i>Package Elements</i>	43
6.2.3	<i>Package Relationships</i>	43
7	References	46

1 Introduction

Model-based requirements management is becoming a key technological area for automotive manufactures and supply chains. The ATESSST project [6] has laid a sound basis for elementary support of model-based requirement management. However, several challenges remain to be tackled to support upcoming needs of the automotive industry.

In particular, this work task will further advance the state-of-the-art of model-based requirements management in the following areas:

- Full support of RIF (Requirements Interchange Format) in EAST-ADL language and tool framework. RIF is a requirements exchange standard created by the German automotive industry. RIF is a powerful mechanism being not only a definition of a structure of specification objects but also of meta-model extensions. In ATESSST, a concept for including RIF into EAST-ADL2 has been developed. In the ATESSST2 project the concept will be worked out in detail and prototypically implemented. As some of the partners are directly involved in RIF development, the dissemination of (expected) RIF extensions will be relatively straightforward. In particular, a prototypical Eclipse plug-in for RIF import in and export from EAST-ADL2 will be developed in the ATESSST2 project.
- EAST-ADL2 already includes requirements and V&V concepts, but they are not yet integrated; also highly adaptable requirements and V&V views do not exist until now. These textual or tabular views, e.g. for complex filtering and tracing, which go across current abstraction levels of the EST-ADL2, will be defined and added to the language infrastructure.
- Manufacturers and suppliers strive to introduce more and more elements of product line engineering into their processes and specification structures. However, there is still no support for product line engineering when it comes to exchange of specifications hampering the recognition and exploitation of synergies between the respective product lines. A concept for manufacturer/supplier specification exchange will be developed that allows for specification exchange with additional product-line-related information.

Requirements engineering support of the EAST-ADL provide for powerful adaptation and exchange mechanisms and in particular include tracing between requirements and tests. Requirements are the input information for any system design effort, and contain the constraints and rules during the entire design effort. To improve dependability and in particular safety, rigorous treatment of requirements is critical. An architecture description language captures the elements that compose the modelled system architecture in a comprehensive way. It is therefore desirable to also investigate requirements, and how they can be associated to the system elements. An evaluation of the support for requirements in EAST ADL is therefore an important input for the further refinement of requirement constructs.

The EAST-ADL specification defines a domain-specific modeling language for automotive systems engineering applications, first defined the EAST-EEA project [6]. Throughout the rest of this document, this language will be referred to as **EAST-ADL**.

The EAST-ADL specification was refined in the ATESSST project, www.atesst.org [7]. Throughout the rest of this document, the refined EAST-ADL language will be referred to as **EAST-ADL2**.

The EAST-ADL2 language is aligned with AUTOSAR language constructs, www.autosar.org [8]. Throughout the rest of this document, AUTOSAR language constructs will be referred to as **AUTOSAR**.

EAST-ADL did not make the distinction between generic and specialized requirements. Some special constructs from EAST-ADL are no longer supported as dedicated entities in the domain model, because it is preferable to simply model the corresponding information in form of generic requirements.

As done for the structure part of EAST-ADL2, the requirements part will be compliant with SysML and UML2. The EAST-ADL2 refines existing concepts whenever possible, and develops new ones otherwise.

The purpose of the meta-classes in the EAST-ADL2 requirements domain meta-model package is to specify rigorously ("formally") the requirements concepts for the automotive domain.

Support for requirement modeling is provided by the EAST-ADL2 on two levels: a generic level and a specialized level where specialized requirement entities are provided which are intended for certain special uses.

2 Evaluation and update suggestions for requirements and V&V concepts

This document is a summary of the current EAST-ADL2 requirement support concepts, and includes only an initial assessment. Below is a preliminary and early list of changes that should be implemented on the language.

2.1 Language modifications

The following is a tentative set of changes that should be considered related to requirements concepts:

- Timing requirements and constructs should be updated to be aligned with TIMMO timing concepts and MARTE
- The ADLRequirement entity needs an attribute for the requirement text, in SysML called “text”.
- SysML is highly relevant and the EAST ADL should be aligned with the relevant SysML concepts such as requirements and V&V. However a re-formulation should be considered so that there is no inheritance in the domain model. The domain model should be sufficiently aligned with SysML to allow the stereotypes in the profile to be specializations of the SysML stereotypes.
- The same reasoning is applicable to MARTE: Domain model concepts should be aligned where possible, but MARTE entities are not part of the domain model.
- The documentation for Traceable Specification may be updated to give examples of its purpose.
- Use cases are currently singular entities. It may be appropriate to add attributes to improve the guidance and allow fine-grained variability. Examples of attributes are
 - Actor (may be introduced as separate metaclass)
 - Primary description
 - Alternative path
 - Exceptions
 - Triggers
 - Precondition
 - Minimal Guarantee
 - Success Guarantee
 - Actions
 - Constraints

2.2 Profile related modifications

EAST-ADL2 has links to the SysML both on the meta model level and profile level. Since EAST-ADL2 was defined in ATESSST, the SysML has evolved and certain modifications have an impact on the EAST-ADL2 UML2 profile.

- Satisfy in SysML is now a dependency rather than a realization. This should also apply to ADLSatisfy

2.3 EAST-ADL2 support for contextual allocation and contextual ownership

EAST-ADL2 has support for contextual allocation and contextual ownership. The EAST-ADL2 context construct is used to achieve a simple and practical way to allocate traceable specifications to a specific EAST-ADL2 model context, and to let this specific model context own its comments and relationships.

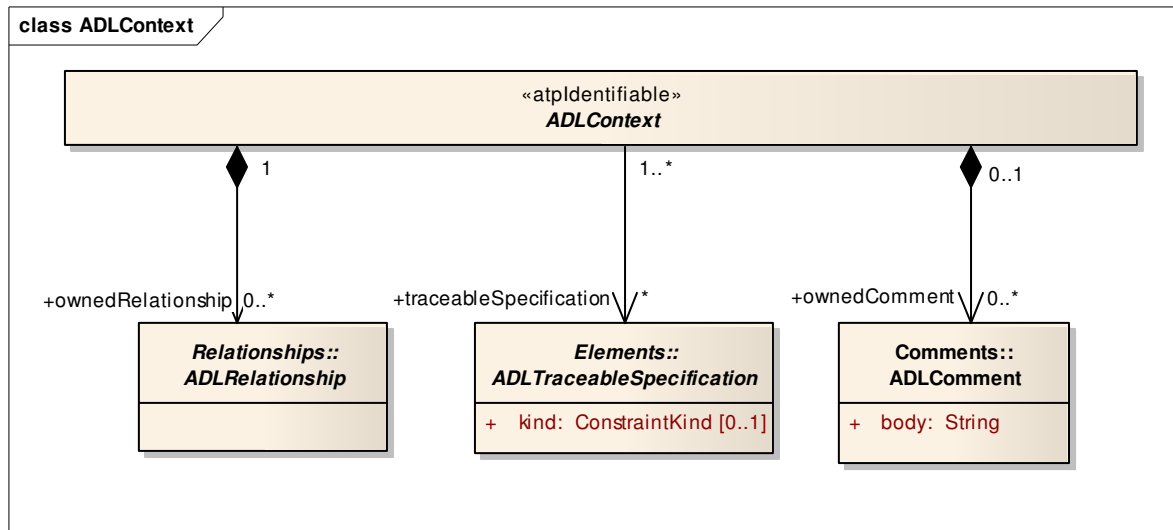


Figure 2. ADLContext and its contained entities – Potential modification

A modified extract from the domain model, the proposed change is the new role name for the association in-between ADLContext and ADLTraceableSpecification. The association in-between the context and its allocated traceable specifications is a relationship, but should be handled explicit to achieve clarity.

2.3.1 Elaboration around definitions

Cambridge dictionary: **allocate**; to give something to someone as their share of a total amount, for them to use in a particular way:

Cambridge dictionary: **satisfy**; the condition/need/requirement to have or provide something that is needed or wanted:

The traceable specification role name *traceableSpecification* is used to allocate traceable specifications to a specific context, as the context share of the total amount, for the context to relate the allocated traceable specification to satisfying and verifying constructs.

Thus the contextually owned satisfy relationship relates allocated traceable specification to satisfying construct/s, and the contextually owned relationship verify relationship relates allocated traceable specification to verifying construct/s.

2.3.2 EAST-ADL2 contextual allocation; allocation of traceable specifications to a specific context

In this activity, traceable specifications for the system have to be allocated to a specific context, e.g. it can be allocated to a specific vehicle feature or functional architecture composition.

This allocation of traceable specifications is the basis for the analyses and design steps taking place on corresponding abstraction levels.

It is necessary to observe that the allocation of traceable specifications to a specific context must fulfill the following criteria:

- Every traceable specification must be allocated to at least one context, ideally exactly to one context.
- Each traceable specification is allocated to the context corresponding to appropriate abstraction level, which makes it possible to meet the traceable specification in its most

appropriate context. Normally, the total of the traceable specifications have to be allocated to various abstraction levels.

- The allocation must be realized in such a manner that it will be possible to prove the fulfillment of the traceable specification by checking the corresponding contextual architecture elements and relationships.

The rationale for this EAST-ADL2 construct is that in many cases, entities such as functions and sensors need to have requirements and other specification elements allocated to the entity. In other cases, the relation between the entity and the related specification element is specific for a certain context: for example a requirement on a sensor is only applicable in certain hardware architecture. The EAST-ADL2 construct for traceable specifications is the super-class for the specification entities of the language. Specializations of traceable specification are for example the EAST-ADL2 requirement construct and use case model.

Requirements can be allocated to features or function architectures, and the features or function architectures can be allocated to vehicles. This would be in-line with a functional development approach, which is to be encouraged for most systems of some complexity.

To make the allocation in EAST-ADL2 more general, all traceable specifications in EAST-ADL2 can be allocated to an EAST-ADL2 context.

One purpose with allocating requirements, and other traceable specifications, to EAST-ADL2 contextual abstractions is to ensure that the appropriate type of specification is stated, at the right abstraction, at a certain stage of development.

2.3.3 EAST-ADL2 contextual ownership; owned relationship and owned comment

Only necessary information should be included in statements in traceable specifications to make them easier to read, understand, and less susceptible to different kinds of mistakes. Unnecessary information should of course just be removed, but in many cases information has a value even though it is not really part of the traceable specification. Such information should be recorded as contextual commented information (comments) owned in the context where the traceable specification is allocated, but separated from for the statement.

Relationships, i.e. how a traceable specification relates to different ADL entities, should be recorded as contextual relation information (relationship) owned in the context where the traceable specification is allocated.

The SysML requirement construct is aware of its target block; this is “disabled” in the EAST-ADL2 requirement construct, using contextual ownership of relationships.

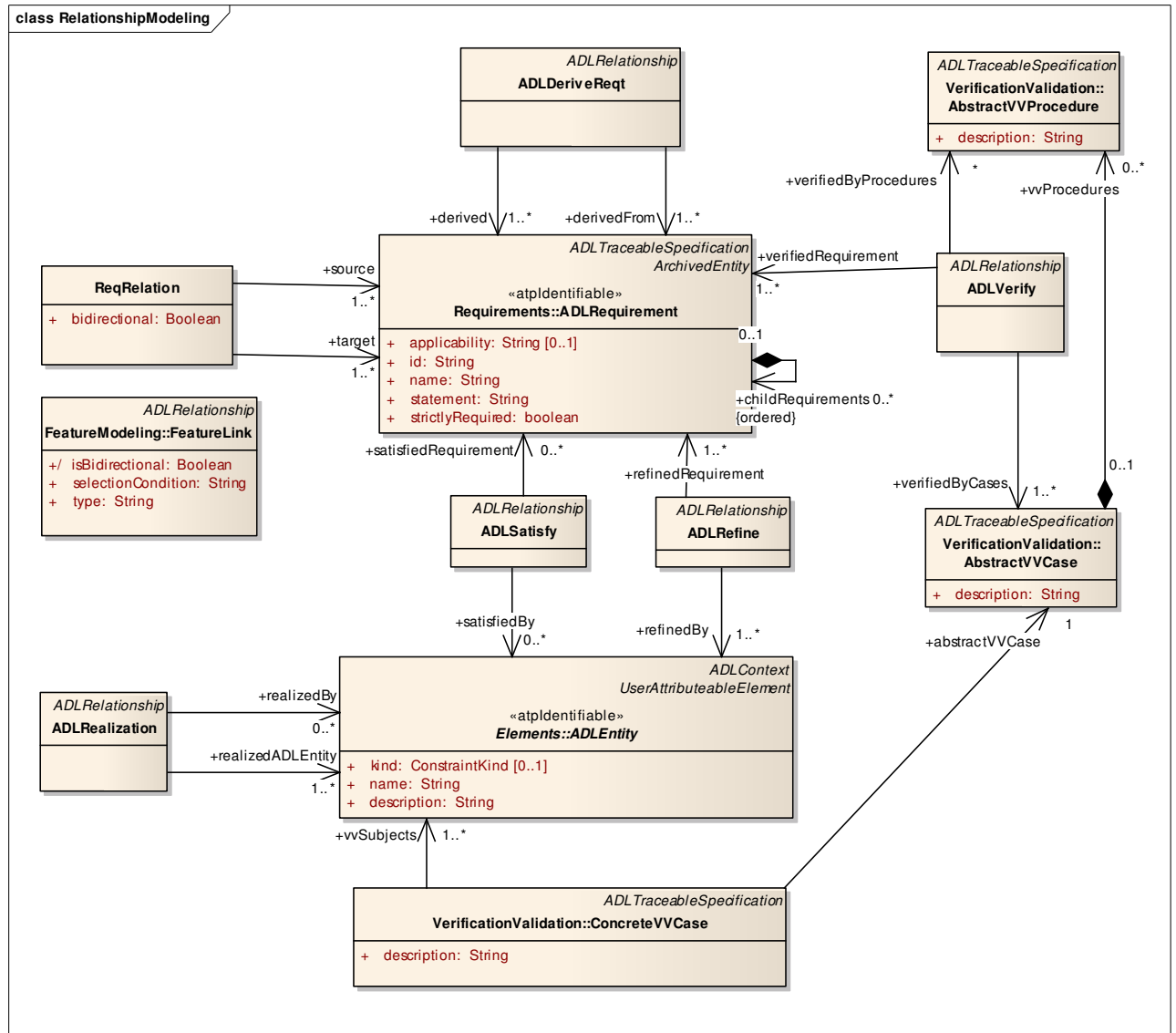


Figure 3. Relationship entities in EAST-ADL2, domain model as of 2009-01-27

Note the specification of the ADLRelationship specializations ADLSatisfy, ADLRefine, ADLVerify, ADLDeriveReq, and ADLRealization. These concepts are implemented as UML Dependencies in the UML profile.

2.3.4 EAST-ADL2 contextual owned satisfy relationship

The EASTADL2 satisfy relationship, ADLSatisfy, is a dependency construct between a traceable specification and a model element that fulfils the traceable specification. It can be the context for the traceable specification, with possible hierarchical owned satisfying constructs, that satisfies the specification.

2.4 Requirement engineering with EAST-ADL2

For requirements engineering, it is important to begin with the requirements on the product and what it should achieve, without a lot of detailed requirements on individual components. Otherwise, there is a risk missing the overall objective or getting lost in a lot of details early in the project. When allocating requirements to EAST-ADL2 contextual abstractions, it will be recognized which

requirements that are high level requirements and which requirements that are detailed requirements.

Another purpose of requirement allocation is to support the process of deriving responsibility from the product (system) into several sub-systems. This will enable parallel development of the sub-systems in an effective way.

Working with requirement on different abstraction levels doesn't mean that the requirement process needs to be strictly ordered, in reality some detailed requirements pop up early. Using EAST-ADL2 requirement abstraction levels, those requirements can be put on the right abstraction level in the structure.

2.4.1 EAST-ADL2 solution-free requirements-handling

Keeping requirements free from solution is very important for producing well reusable requirements. Solution-free requirements are related to the EAST-ADL2 abstraction levels. A requirement on the vehicle level should not rely on any particular system design.

If the system design is known early it can be modeled on the corresponding abstraction levels and the requirements on appropriate abstraction levels can then be derived. There is no contradiction with having EAST-ADL2 solution-free requirements-handling, and starting with the system design before all requirements are known. Keeping requirements and solution separate will clarify and justify the role of requirements.

2.4.2 Formulation of EAST-ADL2 requirements

A basic rule is to not describe how something works, but rather what is actually required. Terms can also be used for statements that are not strictly required, but this is then formally more a wish and not a requirement. It is of course relevant to formulate wishes as well, but it is preferred to also use an attribute to indicate what statement that are requirements and what statements that are wishes.

As far as possible it is recommended to specify one requirement per statement. If several requirements are combined there is a risk for getting complex statement that is difficult to understand. Different interpretations of what is actually required are an unwanted situation.

2.4.3 EAST-ADL2 use-case model

To create a use-case model in EAST-ADL2, best practice from normal UML modeling can be used. A desired improvement is to view use cases as a way of documenting the sources of requirements, and modeling identified requirements separately with traceability to the use cases. Use cases can be considered too informal to use as the only specification of functional requirements on an embedded system.

The introduction in the EAST-ADL2 of the actor meta-class to promote re-uses and harmonization of a defined set of actors, is proposed.

2.4.4 Stakeholders and stakeholder needs in EAST-ADL2

Stakeholders and stakeholder needs are not currently covered by EAST-ADL2. The stakeholder concept is quite similar to actors: A stakeholder has an interest in a system; the stakeholder can lose or gain from the system. An actor interacts actively with the system, which means that the actor is per definition also a stakeholder. It should be OK to model a stakeholder like an actor in EAST-ADL2, but without participating in any use case model, though still having needs/requirements on the system.

Ideally a stakeholder stereotype should be introduced as a generalization to the actor, which is proposed here.

Two meta-classes are added to model stakeholder and stakeholder needs. The proposal is to associate to the problem statement meta-class.

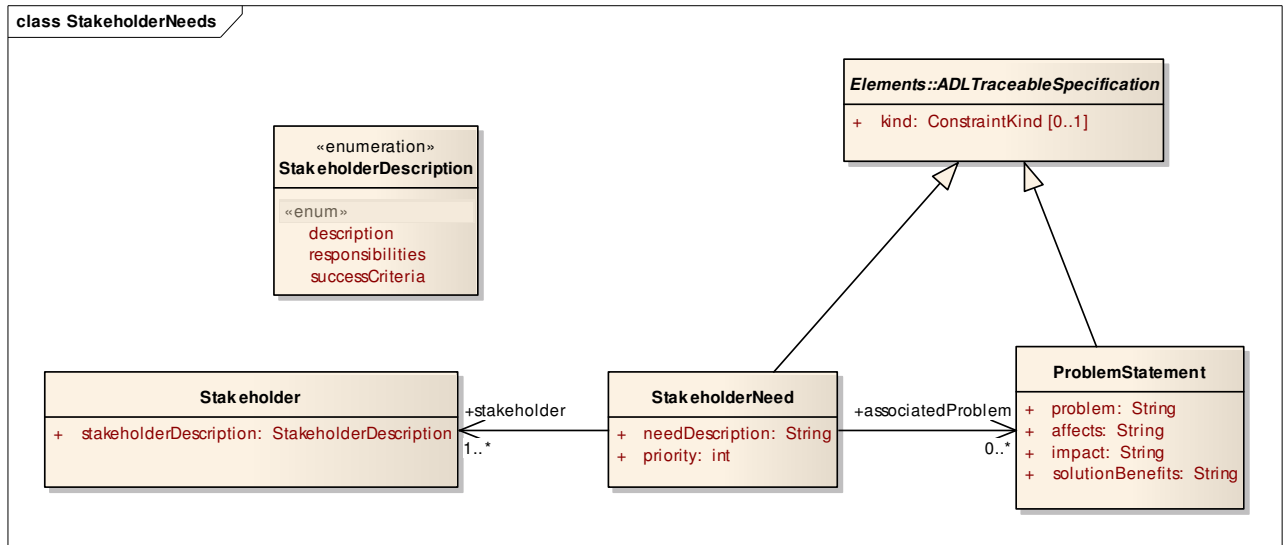


Figure 4. Stakeholder needs – Potential constructs

2.4.5 Use case relation to product features in EAST-ADL2

One concern is how features should relate to use cases. Since the feature level is the highest level of abstraction in EAST-ADL2, the natural choice would be to include use cases on the feature level. This makes sense since use cases and features are roughly on the same level of abstraction. Use cases give the view of how actors interact with the system, whereas features tell what the system provides. This means that a feature can satisfy a use case.

In most cases a feature would be related to one or several use cases, but some features could be independent from use cases. Since they are on the same abstraction level, the same requirements could be associated both to use cases and features, i.e. use cases as the source and features as the satisfying entity.

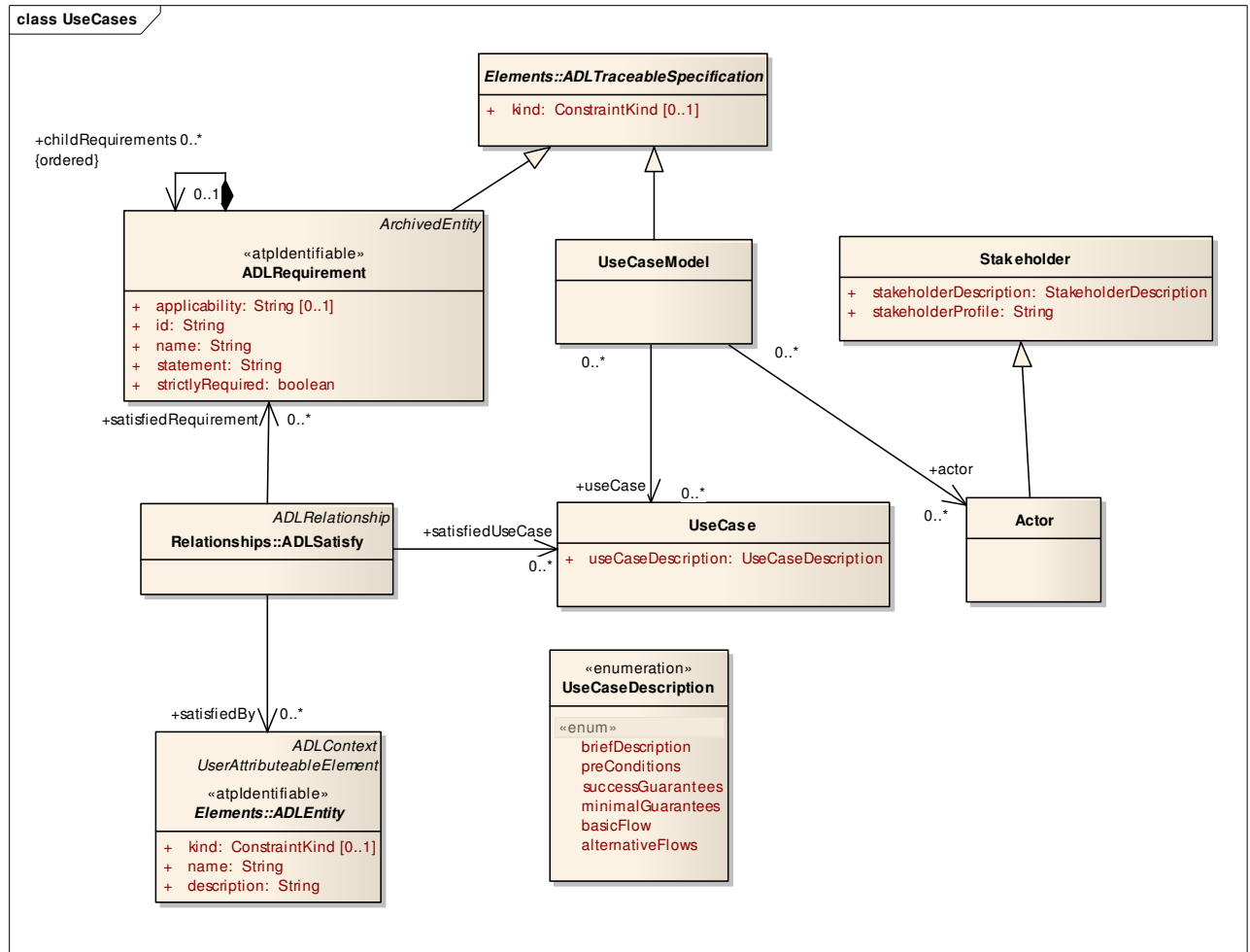


Figure 5. Modelling of use cases – Potential constructs

2.4.6 Using Requirements Interchange Format (RIF) with EAST-ADL2

The import and export of development information to and from the EAST-ADL2 is not accomplished by a direct import or export between EAST-ADL2 and a standard Requirement Engineering tool. Instead, the *Requirements Interchange Format* (RIF) is employed as an intermediate representation which may serve as a uniform link to many different legacy Requirement Engineering tools.

RIF was introduced by the “Herstellerinitiative Softwaretechnik” (HIS), which is a joint effort of the German automotive manufacturers Audi, BMW, Daimler, Porsche, and Volkswagen to achieve joint standards in the field of automotive software engineering, in particular standardized software modules, process maturity levels, software test, software tools and programming of control units.

2.4.6.1 Rationale for Using RIF

Before going into the details, a brief introduction of the strength of the RIF format, which are of particular interest in the context of the ATESSST2 project:, is presented:

- **Flexibility** The basic structure of a RIF specification is extremely flexible: it mainly consists of – optionally inter-related or grouped – SpecObjects of various SpecTypes, which are each provided with a number of AttributeDefinitions.
- **Open Standard.** The specification of RIF is freely available over the web and is further maintained under the auspices of the “Herstellerinitiative Softwaretechnik” (HIS).
- **RIF is maturing** More and more tool vendors provide an import and export from/to RIF for their tools. In addition, the RIF format has drawn during the last two years increasing attention from practitioners and researchers in the field of requirements engineering.

In particular, a requirement in EAST-ADL2 is a very flexible element which mainly consists of project-specific attributes. There are special cases of requirements with predefined attributes and relations, but they can be seen as a special case of a flexible requirement. This matches well the flexible structure of RIF with SpecTypes and their AttributeDefinitions.

Other important features and characteristics of RIF, such as its support for access rights management with AccessPolicy objects, can be found in the RIF specification document.

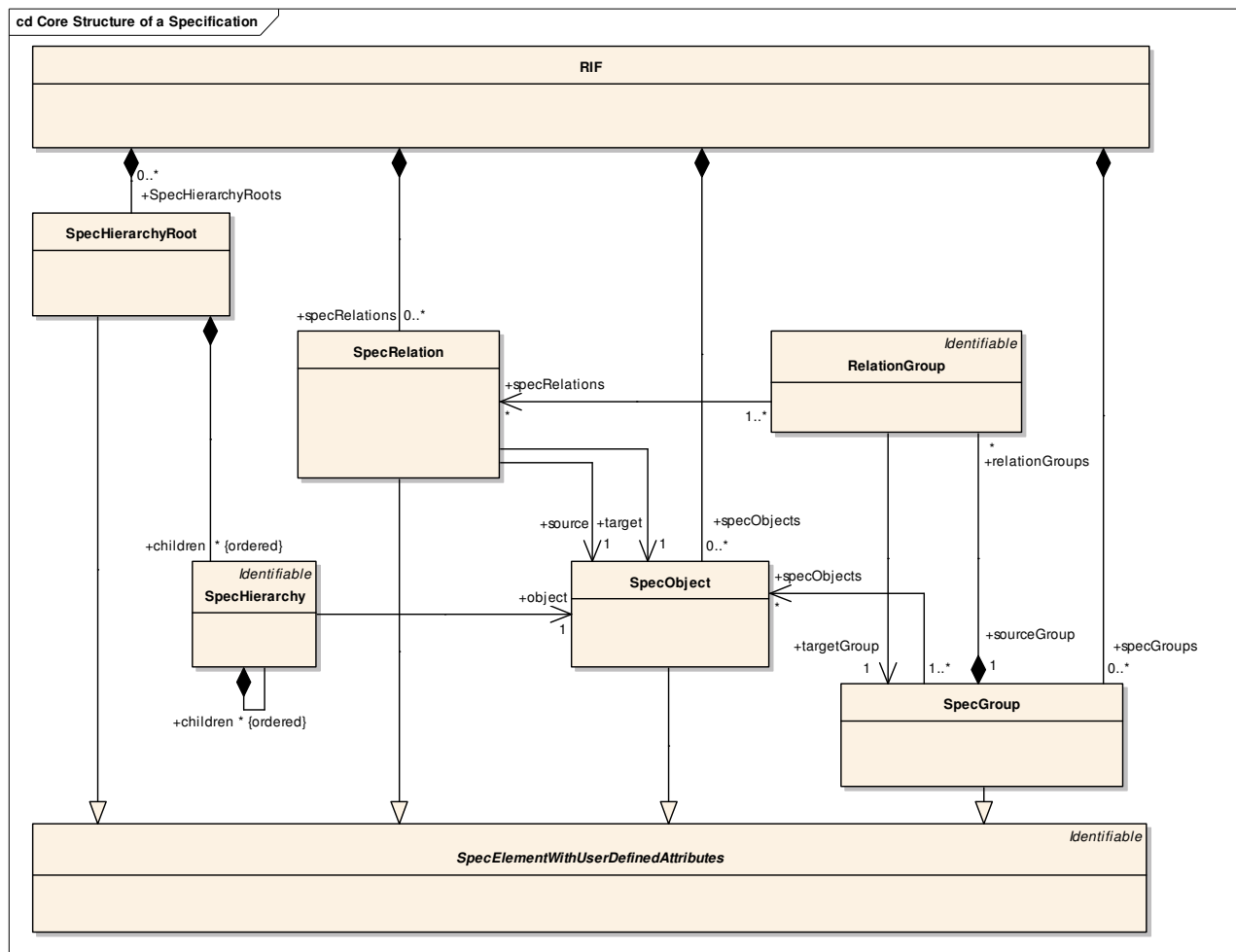


Figure 6. RIF metamodel for SpecObject

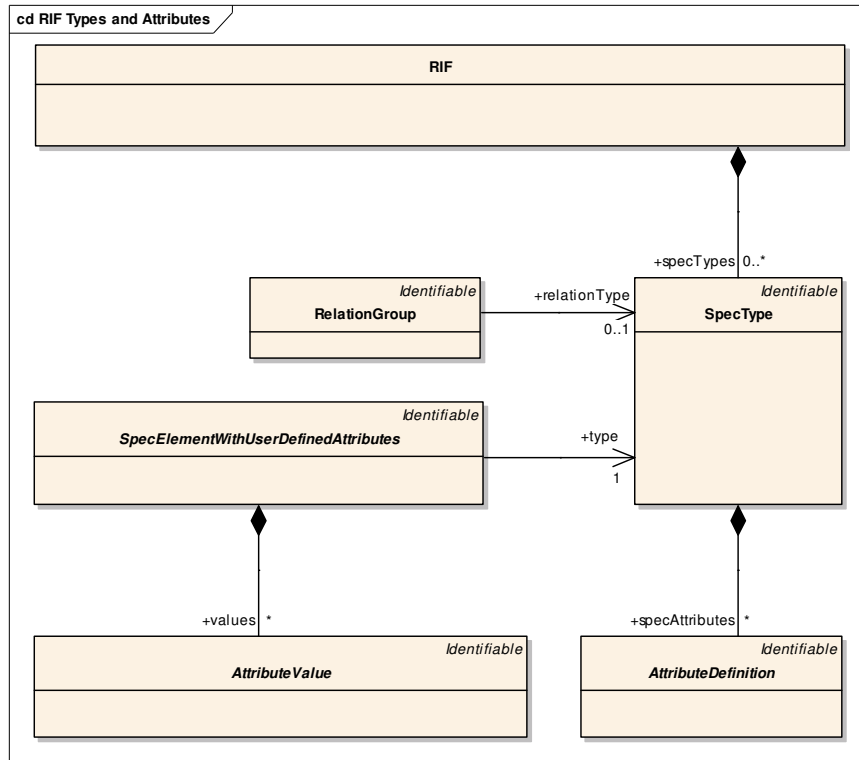


Figure 7. RIF Metamodel for types and attributes

2.4.6.2 RIF Import

When importing RIF content into an EAST-ADL model, several scenarios may occur. For each of them we will list the possible strategies for a tool a user.

1. Perfect match
(i.e. for all SpecTypes and AttributeDefinitions in RIF, there is a corresponding EAST-ADL2 entity or attribute)
2. Missing attribute in EAST-ADL2 for a RIF AttributeDefinition
3. Missing entity in EAST-ADL2 for a RIF SpecType
4. Lack of or inappropriate information on RIF side
5. Fundamental semantic mismatch
6. Combinations of the above cases

In the remainder of this section each of these cases will be looked at in detail.

Case 1: Perfect match

Assumed is that the information on the RIF side perfectly corresponds to EAST-ADL2 entities, but a mapping needs to be provided between the SpecTypes and AttributeDefinitions in RIF, and the EAST-ADL2 entities and their attributes.

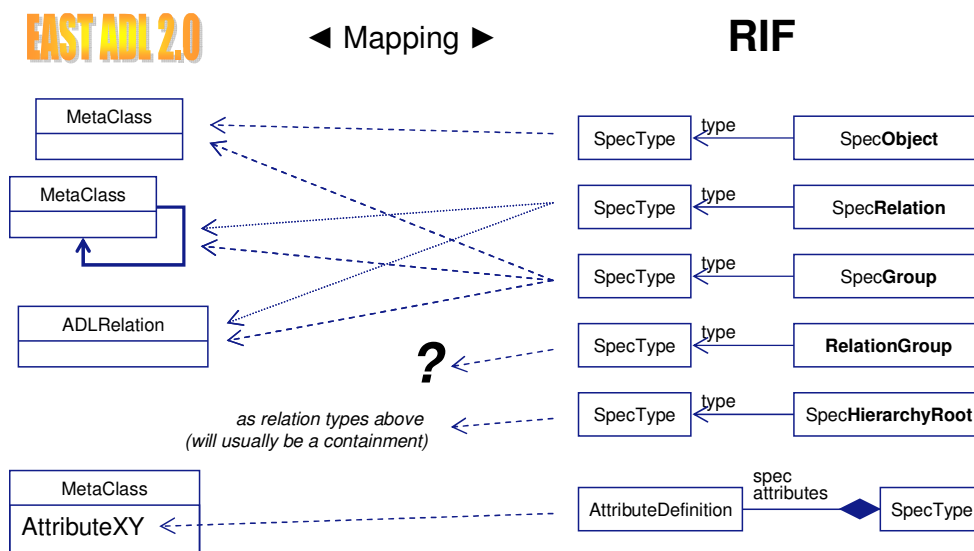


Figure 8. Mapping between EAST-ADL2 entities and RIF

The right side above shows several cases of type and attributes definition in RIF. The cases of SpecType definition differ in that each time the SpecType is used for different sub-types of SpecElementWithUserDefinedAttributes, i.e. SpecObject, SpecRelation, SpecGroup, and SpecHierarchyRoot (highlighted in the figure with bold face font). To simplify the following discussion, a SpecType, which is used as the type of a SpecObject, is called an “object type”, one that is used as the type of a SpecRelation a “relation type”, and so on.

In practice, a single SpecType may be used to type objects of more than one of these classes, i.e. it is possible to use a single SpecType as the type of one or more SpecObjects and at the same time one or more SpecRelations. For the import filter this means that it must be possible to distinguish these cases, i.e. it must be possible to provide a different mapping to EAST-ADL2 entities for the same SpecType depending on whether it is used as the type of a SpecObject or SpecRelation, for example. With the above abbreviation, a single SpecType may at the same time be an object type and a relation type.

Object types are relatively easy in that they are always mapped to a meta-class in the EAST-ADL2 domain model (remember that here the “perfect match” case is assumed).

Relation types are slightly more intricate because two different techniques are used in the EAST-ADL2 domain model to allow relating entities in the ADL:

1. Plain Associations
2. subclasses of the meta-class ADLRelation

When mapping RIF SpecTypes to the EAST-ADL2 these two cases can occur and the import filter has to be able to cope with them.

Group types are also a bit intricate because outside the scope of requirements entities, no generic grouping mechanism is provided in the EAST-ADL2. Since we are still assuming the perfect match case, the information captured within a SpecGroup of that SpecType must have a corresponding entity or attribute in the EAST-ADL2. All kinds of entities, associations and attributes are conceivable as a target for this information. Therefore the import filter must allow mapping groups accordingly.

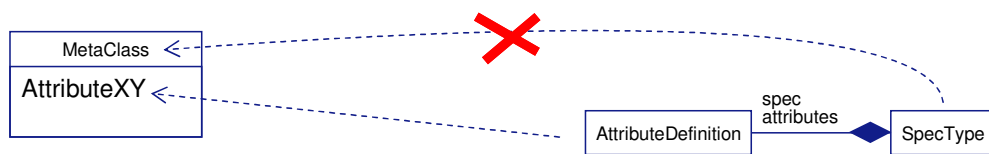
Relation-group types are particularly problematic. Not only do they lack a corresponding generic concept in the EAST-ADL2 but there are also not really any concrete entities in the ADL that may be a reasonable target for importing RIF relation groups. In addition, they are most often used in a

very RIF or target-tool specific manner (for example, relation groups are used to mimic DOORS link modules in RIF). Therefore, there usually won't be a "perfect match" case with respect to relation-group but instead a "fundamental semantic match" (see below).

Hierarchy-root types can be mapped quite easily to EAST-ADL2. At first sight this concepts seem quite odd and incompatible to the EAST-ADL2. However, a SpecHierarchyRoot, together with its child SpecHierarchy objects, simply defines a containment hierarchy on the SpecObjects (with the specialty that a single SpecObject may be part of several orthogonal containment hierarchies, which is forbidden with the usual interpretation of containment). And containment is nothing else but a special kind of association or, in the terms of RIF, a relation. Consequently, the above remarks on relation types equally apply here. The only exception is that in the case of hierarchy-root types the target will most often be a plain containment association. A difficulty that may arise here is that containment hierarchies are often made up of objects of various types; in that case, the containment relation will need to be mapped to different associations on the EAST-ADL2 side, which must be supported by the mapping definition and the import filter.

Attribute definitions are mapped to attributes of EAST-ADL2 entities.

Note that an AttributeDefinition and its corresponding attribute on the EAST-ADL2 side to which it is mapped need not have containers that correspond (and are mapped) to each other, as illustrated in picture below. In other words, the containing SpecType of an AttributeDefinition may be mapped to another class than the AttributeDefinition's corresponding attributes containing meta-class.



This becomes significant when mapping the AttributeDefinitions of relation types to EAST-ADL2. The problem at this point is that in RIF attributes can be defined for all objects of type SpecElementWithUserDefinedAttributes. This becomes problematic if, for example, a relation type is mapped to a plain association and this relation type has AttributeDefinitions, because a plain association can't have attributes. In this case it is important that the attribute from the RIF side can be mapped to an attribute of some other EAST-ADL2 entity.

So far, it is assumed that the information on the RIF side directly matches entity, association or attribute on the EAST-ADL2 side.

Case 2: Missing Attribute

First of all, when mapping an AttributeDefinition from RIF, the problem is identified of not finding an attribute of an EAST-ADL2 entity which constitutes a reasonable match. Thus, some information on the RIF side exists for which there is no predefined slot in the EAST-ADL2, which may occur especially in the case of project- or company-specific meta-information.

Fortunately, we above already introduced a concept to solve this problem: user attributes. In fact, they were added into the EAST-ADL2 as a global concept, applicable beyond requirements entities, precisely for this reason.

Thus, solving this mismatch is simple. For each attribute definition in RIF that does not have a correspondence in the EAST-ADL2, an additional UserAttributeDefinition is introduced for the EAST-ADL2 entity to which the SpecType containing the AttributeDefinition is mapped. If there is no such entity (e.g. in case of relation types which are mapped to plain associations, see above) then the user attribute is defined for some other, related entity.

Case 3: Missing entity in EAST-ADL2 for a RIF SpecType

When an entire entity is missing on the EAST-ADL2 side instead of only an attribute, then the possibility would be needed to add a custom entity to the EAST-ADL2, something like a “user entity”. In principle, this would not be much different from allowing user attributes. However, if such a concept was consequently implemented into the EAST-ADL2, this would lead to several significant downsides:

- It would add a great deal of additional complexity to the language.
- With such a concept, we would in fact mimic or reuse UML2 stereotypes.
- All tools that want to fully support the EAST-ADL2 would have to provide an implementation for this extension mechanism.
- Reusing existing implementations of such an extension mechanism from standard MDA frameworks is not feasible at present (immature, many open issues, ...).
- Offering too much flexibility would conflict the overall objective of EAST-ADL to provide a standardized and unified conception of automotive software development.

Considering these negative effects, it was deemed preferable to not add such an extensive customization mechanism and instead accept this limitation.

This decision was further encouraged by the fact that in the case of missing entities some quite feasible workarounds are available. First, the surplus information on the RIF side can be stored in user attributes attached to some related element. This solution will already serve well in many cases and has the advantage of storing the additional information close to elements to which it is related. Second, if that fails, it is always possible to just do on the EAST-ADL2 side what was actually done in RIF: storing some information which is not a requirement in form of a requirement specification. Precisely speaking, we define a `UAElementType` for the missing entity and store the surplus information in an instance of the EAST-ADL2 Requirement entity, typed by that `UAElementType`.

Case 4: Lack of or inappropriate information on RIF side

When importing information from RIF into the EAST-ADL2, it is always possible that some information which is necessary from the EAST-ADL2 perspective is missing or that the information provided is in some way inappropriate when interpreted in terms of the EAST-ADL2.

At first this does not seem to pose a problem. Even without considering imports, there will always be times during intermediate stages of modeling at which at least some of the model's entities are incompletely defined. However, when taking into account that in the case of an import the source data can be of a more or less orthogonal structure, this may lead to invalid EAST-ADL2 models, particularly due to

- incompleteness
- inconsistencies
- violations of EAST-ADL2 domain model constraints
- ...

There is no technical means to avoid such situations. Instead, a flexible import mechanism must allow the user to manually resolve the emerging violations and conflicts. As a consequence, we can draw the following conclusion: Permitting the import of data that was not originally intended for EAST-ADL2 requires that the import filter as well as the EAST-ADL2 tool used to view and edit the imported data must be able to cope with invalid EAST-ADL2 models, in order to present an invalid import to the user and allow him to manually correct the model.

Case 5: Fundamental semantic mismatch

Beyond the rather manageable mismatches covered by the previous three cases, it is of course possible that the semantic meaning or the structuring of the data on the RIF side is fundamentally irreconcilable with the semantic and/or structure of the related EAST-ADL2 elements.

To illustrate this a bit further, let us briefly consider two examples:

- The RIF file to be imported contains information on the system design which is completely incompatible with FDA entities of EAST-ADL2.
- On the RIF side only a few coarse-grained types were used to characterize the SpecObjects. Then it is likely that we would need to consider related elements or the concrete value of attributes in order to decide on the correct target entity on the EAST-ADL2 side (e.g. the correct EAST-ADL2 type depends on whether a certain SpecObject is contained in a certain SpecGroup or a SpecGroup of a certain SpecType).

Such semantic clashes are a principal problem when relating models of different approaches to each other. It cannot be averted on a conceptual or technical level and trying to do so would unnecessarily and unprofitably complicate the language. But of course it is always possible to revert to the workaround presented for case 4 above to simply import the information as EAST-ADL2 requirements.

2.4.6.3 RIF Export (Evaluation compared with EAST-ADL2)

The export filter must provide similar means as those just described for the import filter and many considerations from above apply to it correspondingly. Instead of being able to interpret the orthogonally structured information on the RIF side and map it to EAST-ADL2 constructs, the export filter must be able to generate such information structures. For obvious practical reasons the export filter should use the same mapping description as the import filter to minimize the effort for the end-user.

However, an additional issue is to be mentioned here. It is also related to the import filter but its significance becomes more evident when exporting information. The problem comes up when the exported information is intended to be again imported in some other, standard RE tool, such as Telelogic DOORS, see figure below.



Such a practice is encumbered with the following obstacle: it is difficult to achieve a feasible presentation (i.e. suitable for reading and editing) in the target tool, because each target tool's specific interpretation and application of RIF would have to be taken into account to precisely determine the final structure and presentation in the target tool. Unfortunately there are numerous such tool-specific peculiarities (for example, SpecGroups are often used to further characterize SpecObjects in addition to their SpecType).

As a consequence, the export filter would need to take into account which standard RE tool is being targeted for external viewing and editing of the EAST-ADL2 data. However, this is an issue pertinent to all RIF import / export filters and not specific to the EAST-ADL2 case.

2.4.6.4 Further Topics

Further topics in connection with the plugin include:

- Implementation Options: When implementing an import/export as described above, various existing methodologies and frameworks may be considered as a starting point, for example:
 - model transformation techniques and frameworks
 - the EMF Mapping library (part of the Eclipse Modeling Framework)
 - ...
- Advanced Capabilities of RIF: The RIF format has a few advanced features which may also be interesting from the EAST-ADL2 point of view. For example, RIF provides support for so called AccessPolicies which allow tagging the data in a RIF file with certain read and write permissions in order to highlight what may be changed by the person or company that the data is sent to. It would be desirable to be able (1) to generate such access permissions during export from EAST-ADL2 to RIF and (2) to import such permissions into the EAST-ADL2 and to store them for later export.

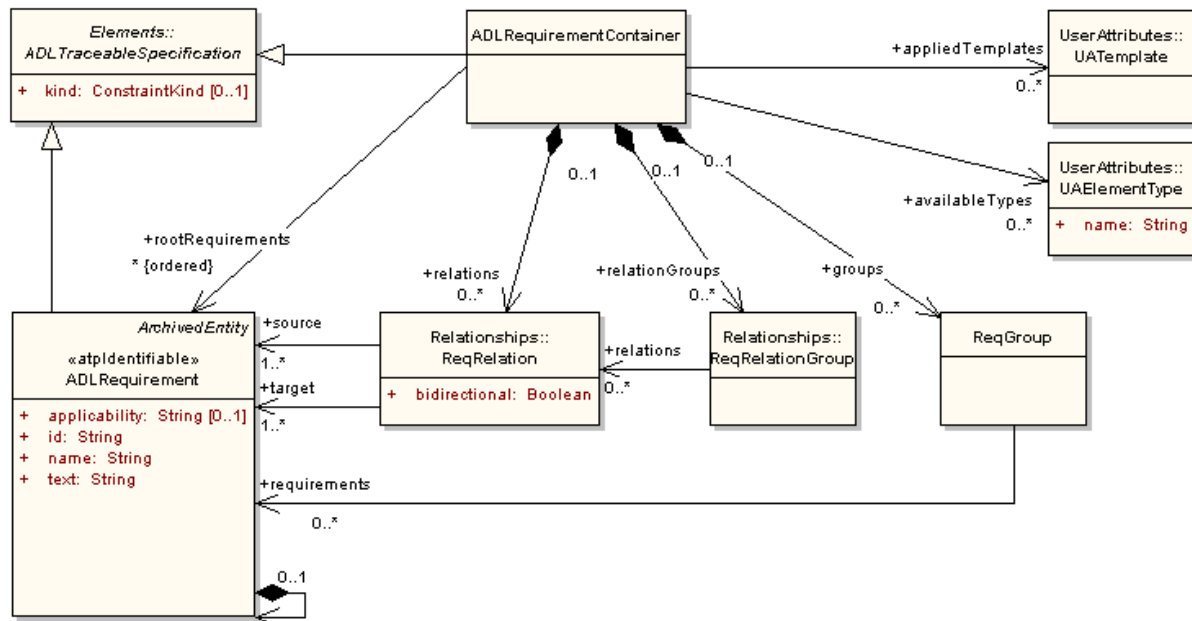


Figure 9. Part of the current requirement domain model as of 2009-01-27.

Supplementary constructs for organizing the Requirements. An additional concept ReqRelation is available between requirements, ReqGroup can be used to group requirements. It should be clarified in ATESST2 whether ReqRelation is an ADLRelationship, whether ADLRequirementContainer is an ADLContext. Also the use of user defined attributes, templates, and types should be clarified. Why are for instance the UATemplate and UAElementType referred to from ADLRequirementContainer, user attribute support should be a quite general concept applicable to several domain model concepts. Note that several options are possible on how to organize requirements hierarchically. The current list of containments includes ADLRequirementContainer-ReqRelation, ADLRequirementContainer-ReqRelationGroup, ADLRequirementContainer-ReqGroup, and ADLRequirement-ADLRequirement. As mentioned in the main text the separation of the requirements and the solution is a main guideline.

2.5 Traditional requirement process as input for update suggestions in EAST-ADL2

A traditional development process is divided in several disciplines e.g. requirements, analysis/design, test etc. Since WT3.1 is about requirements modeling, the focus will be almost entirely on the requirements discipline. Testing and configuration management may be looked at to assess how to link requirement modeling with testing and configuration management of requirements.

A traditional development process is divided in phases, each which can consist of one or many iterations. The requirements discipline is however mainly performed in the early phases. The remainder of the work in the requirements discipline is mainly management of changing requirements.

To adapt the requirements discipline to models, sometimes some simplifications is made to prevent models becoming too complicated. The relationship between different pieces of information needs to be more strictly defined in a model, and overlaps and redundancy in information should be avoided to keep models simple.

All project-related information is excluded from the model, since the philosophy is to only model the product itself, and the models is wanted to be generally reusable in several projects. Project related information is assumed to be recorded elsewhere, in documents or models.

2.5.1 Business opportunity, problem statement and product positioning in EAST-ADL2

Business opportunity, problem statement and product positioning could be modeled with three new meta-classes in EAST-ADL2. Keeping this information in the model will give the opportunity to establish traceability from artifacts created later, for example to provide rationales to design decisions or trade-off analysis.

Note that positioning is assumed to belong to a particular context, typically a system, but also for a smaller part of a system.

Also, the problem statement could be extended with further modeling of dependencies between different problems and deduction of root problems.

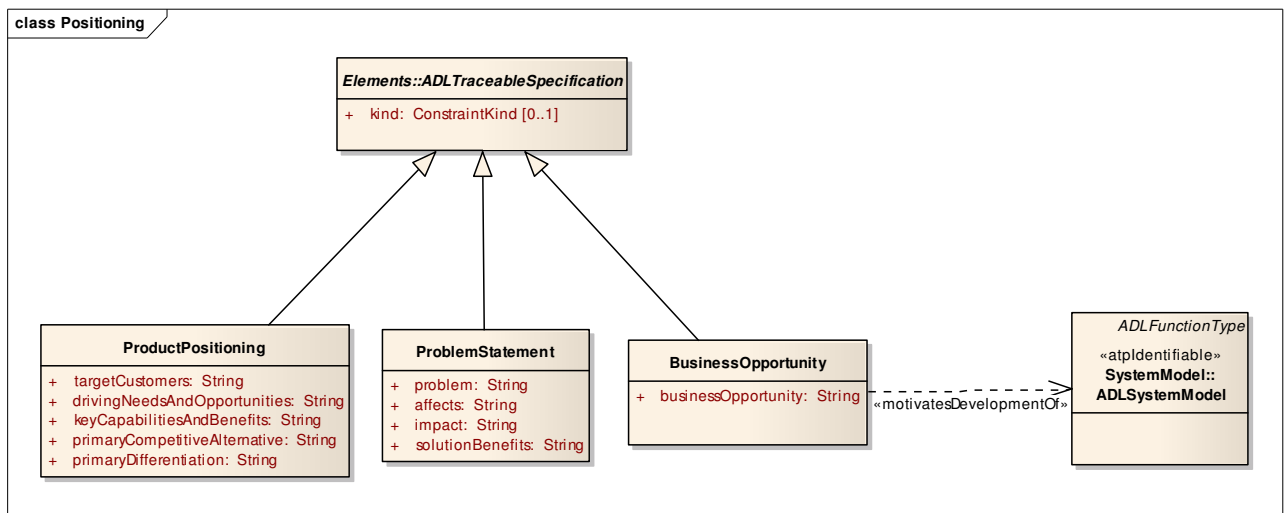


Figure 10. Entities related to modelling information in early phases – Potential constructs

2.6 EAST-ADL2 support for requirement review

Requirement review is an important tool to ensure the quality of requirements and to discover potential faults early. The review is also a good way to learn and improve how to write requirements.

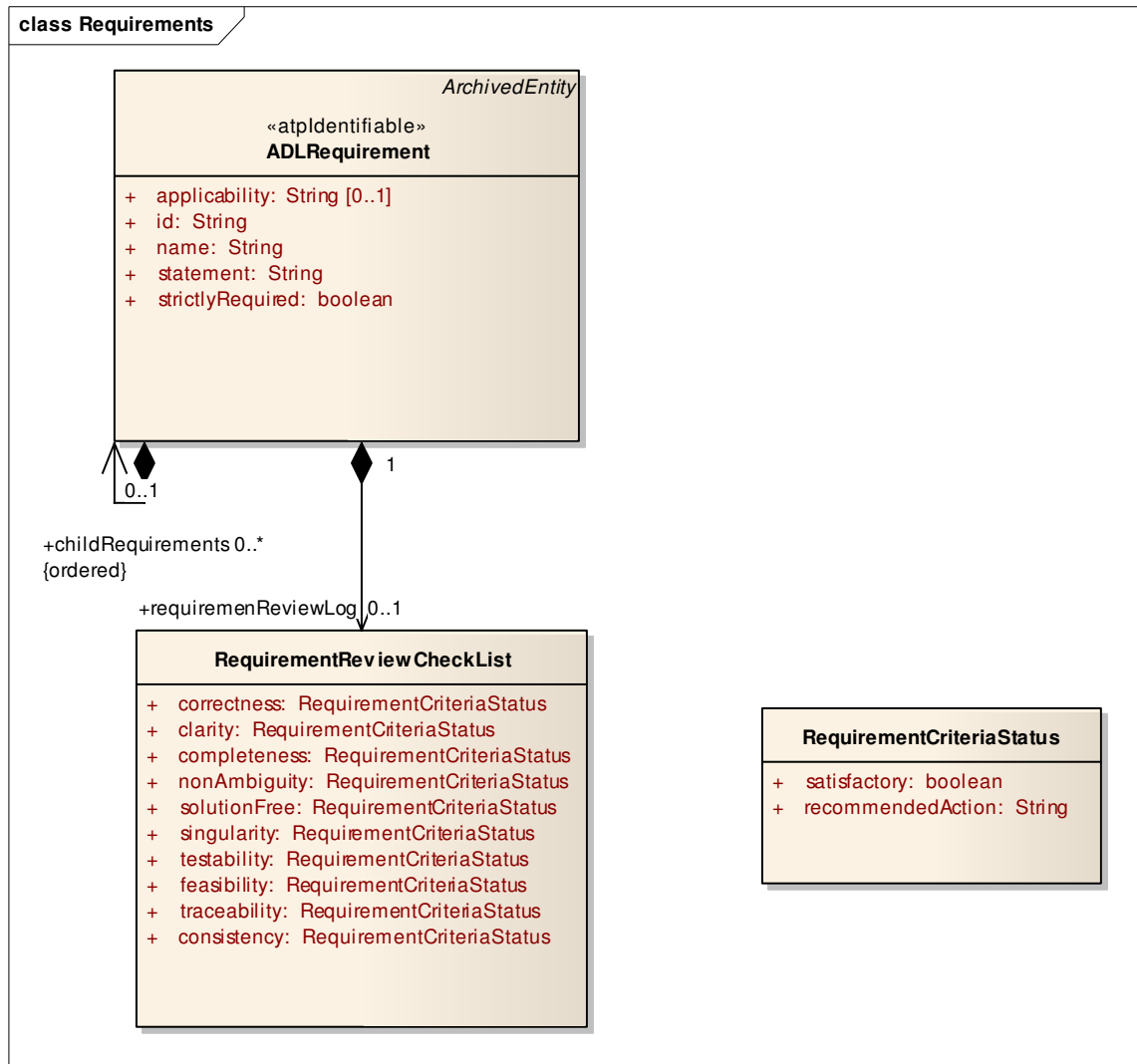


Figure 11. Support for requirement review – Potential constructs

2.6.1 Requirement review checklist in EAST-ADL2

In the check list for requirement review a set of defined criteria can be checked for each requirement statement. With this approach, a wide range of potential flaws can be detected easily and quickly. When a potential flaw is detected, an action is proposed to revise the requirement statement. The following attributes is a set of recommended criteria:

- **correctness** indicates if the requirement statement is valid and free from errors of fact.
- **clarity** indicates if the requirement is readily understandable without semantic analysis.
- **completeness** indicates if the requirement statement contains all information necessary; including constraints and conditions, to ensure that the implementation will fulfill the full intent.
- **non-Ambiguity** indicates that there is only one semantic meaning of the requirement.
- **solution-Free** indicates that the requirement is stated in a solution-free manner.

- **singularity** indicates that the statement contains one requirement only.
- **testability** indicates that a finite and objective procedure to verify that the requirement has been satisfied exists.
- **feasibility** indicates that the requirement can be satisfied using available state-of-art, within natural physical constraints, and absolute project constraints.
- **traceability** indicates that the requirement can be traced to cause, motivation or rationale, plus the stakeholder that the requirement originates from.
- **consistency** indicates that the requirement is free from conflicts with any other requirement.

2.7 Relation to non-software RM and QA Models

In the automotive industry, both requirement management and quality assurance activities (RM and QA) extend far beyond software development, comprising the entire lifecycle of the complete vehicle.

The multitude of process contexts and domains has led to concepts and tool support (e.g. DOORS, QualityCenter) that expose some infrastructure ruptures with respect to a "model-based" approach such as the EAST-ADL.

- RM/QA tools are used for a wide range of activities in requirements engineering (risk analysis, cost calculation, test planning, test execution, test evaluation....). The EAST ADL has to decide which of these information to represent in the language. In the ATESSST project, a concept of flexible "user attributes" has been introduced in order to model such information.
- RM/QA tools have very flexible configuration possibilities, including metamodel adaptations. On the one hand, there exist well-known public and company internal templates for e.g. requirements, but development projects often end up using an extended subset of these templates. The EAST-ADL1.0 had a single fixed meta-model, which is not surprising considering the goal of standardization especially on the Design and implementation level. However, this was not flexible enough to represent RM/QA information in the EAST-ADL. Therefore, in the ATESSST project, a concept of "user attributes" was introduced to allow adapting the meta-model to represent such information. However, this concept needs to be evaluated and also prototypically implemented.
- RM/QA tools are tightly coupled to databases and administer information elements (including history, user rights, versions etc.) on a fine-grained basis (e.g. individual requirements or groups of requirements). Tools for model-based development (e.g ML/SL) are file-based and administer on the level of entire models. The EAST-ADL has reflected this by introducing "infrastructure concepts" (e.g. versioning, history) and allows to require these on the level of individual elements (e.g. requirements). However, while this is conceptually clean, it does not really solve the practical problem of different tool infrastructures.

2.8 Information Exchange with RM/QA tools

The EAST-ADL aims to support information exchange scenarios with requirements and quality assurance tools. These tools support a fine-grained administration of elements. EAST-ADL does not intend not replace the concepts in these tools but provide a usable synchronisation of information from these tools to its "model-based" approach.

The minimum for a usable synchronisation is to map elements from external tools to the EAST-ADL meta-model. Given the EAST-ADL as it is, this might imply that some information will be

collapsed, making it hard to transfer back changes. For this reason the following scenario was identified as a minimum level of integration:

- (in a roundtrip manner): To create and edit requirements and test cases (and links) in the EAST-ADL and export this information to RM and QA tools

By supporting meta-model adaptations, information outside the basic EAST-ADL elements can be represented. The following two scenarios relate to this:

- (in a roundtrip manner): To create and edit requirements and test cases (and links) in the EAST-ADL and export this information to RM and QA tools.
- (in a roundtrip manner): To import information from RM and QA tools, and, where allowed, modify it in the EAST-ADL, and export this information to RM and QA tools.

In ATESSST, the RIF concepts were studied to see how the RM/QA tool exchange could be supported. Based on this, user defined attributes were defined as a adaptation possibility for the user. This way, basic compliance with external RM and QA tools can be achieved, as well as process and organization specific information on each model entity.

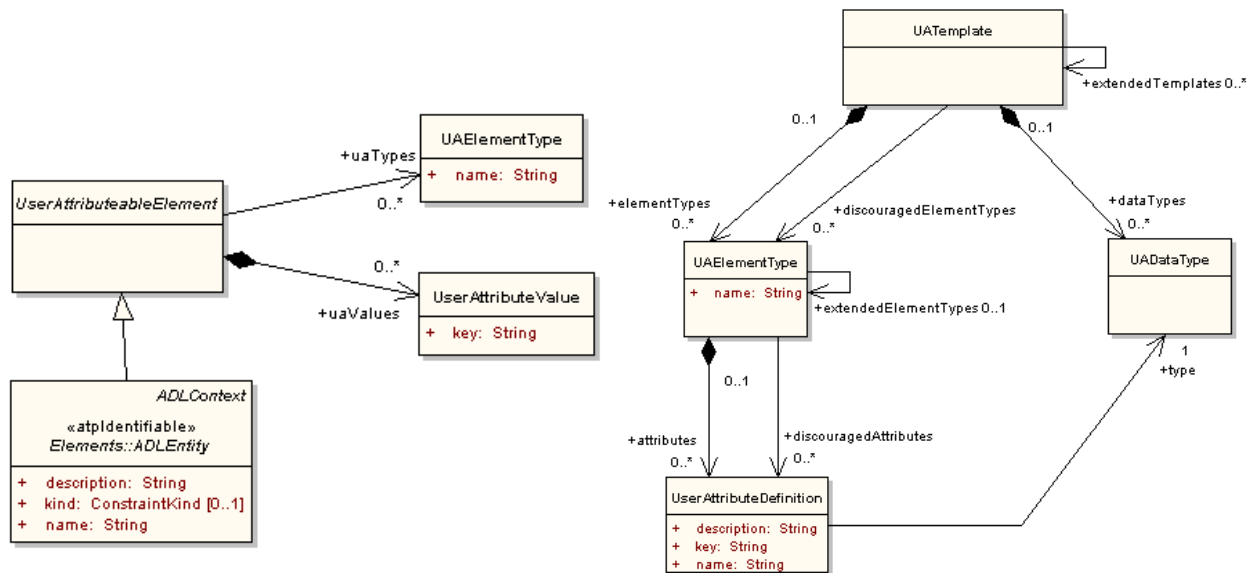


Figure 12. User attributes in the domain model as of 2009-01-27.

2.9 Analysis and Evaluation of EAST-ADL Support for Timing Requirements

One of the goals for the ATESSST2 project is to be compliant with AUTOSAR. The current release 3.1 of the AUTOSAR standard includes only some simple means to describe timing, e.g. in the Timing Model of the VFB Specification [11]. However, this Timing Model is not part of the official standard. A timing team working in the AUTOSAR consortium, which develops a more sophisticated Timing Model for AUTOSAR has recently finished its concept definition phase. This Timing Model will be part of the upcoming AUTOSAR release 4.0. The AUTOSAR Timing Team has been closely working together with partners from the TIMMO project [16]. The objective of TIMMO is to develop a formal description language and methodology to deal with timing information during the development of automotive real-time systems. By means of the close cooperation between the groups, first results of TIMMO have been transferred to the AUTOSAR standard. TIMMO results, i.e. the description of timing constraints and timing properties, will also be included in the next version of the EAST-ADL2 developed by ATESSST2.

In ATESSST, the intention was to achieve a tight integration with MARTE [15] regarding (among other things) timing constructs. As a result there is a preliminary definition for the relation between timing concepts in EAST-ADL2 and MARTE. This work is further refined in the ADAMS project.

The ATESSST result contained several timing constructs, for example, there is a requirement entity called `TimingRestriction` that allows to define bounds on system timing attributes like end-to-end delays, periods, etc. A `TimingRestriction` can specify an upper bound, a lower bound, and/or a nominal value. Furthermore, the EAST-ADL2 defines the attributes `ExecutionTime` and `Period`. These attributes are not requirement entities, but rather properties of the corresponding entities (e.g. functions).

In AUTOSAR VFB specification [11] there is a defined a number of timing attributes of communication patterns. In general attributes in AUTOSAR can be either model-attributes or implementation attributes. The former specify the intention of the developer as they are assigned to SW component descriptions. They are also independent of deployment scenario such as network topology or SWC allocation to ECU platform. The latter defines how a specific implementation is realized and depend on platform, bus etc.

For sender-receiver communication the set of model attributes on a connector includes `TRANSFER_TIME` and `JITTER`. `TRANSFER_TIME` specifies an end-to-end requirement on how much time it may take to transfer data from the sending component to the receiving component. Its value is either of three: `deadline` (max age), `exact_transfer_time` (must be defined together with `JITTER` attribute, or no time constraint set. The `JITTER` attribute is either a maximum value, or no constraint set. For both these attributes are defined as being on AUTOSAR meta level I (Instance and not Type), abstraction level M (Model and not Implementation), and abstraction type R (Requirement and not Behaviour).

The timing model of EAST-ADL2 can be seen to cover what is needed for sender-receiver communication in AUTOSAR as defined in [13]. However, the discussions in the AUTOSAR consortium pointed out that an ADL must be capable of expressing more than this. The goal of the TIMMO project [16] is exactly to close this gap. TIMMO will, among other things, define a Timing Augmented Description Language (TADL), based on AUTOSAR, EAST-ADL2, and MARTE. A review of the AUTOSAR, EAST-ADL2, and MARTE frameworks by the TIMMO partners revealed that there are timing concepts in one or more of the frameworks where the semantics are unclear, mutually incompatible, or lacking expressiveness at certain design levels. For this reason, TIMMO initiated the development of a new “stand-alone” timing semantics. The new semantics is based on an object-oriented model similar to that of AUTOSAR and EAST-ADL2, but with a significantly more concise formalism. A first draft of the TADL is already available in [14]. It is planned to extend the timing specification part of EAST-ADL2 with the results of TIMMO.

2.9.1 TIMMO Timing Concepts

Since the definition of EAST-ADL2, the ITEA TIMMO (TIMing MODel) project has continued working on defining timing properties and requirements in the context of AUTOSAR and EAST-ADL2. The timing concept of TIMMO TADL (Timing Augmented Description Language) is based on the notion of events, event chains and constraints.

Events are defined for the various occurrences that can be observed, for example the transmission or arrival of data on a port.

An event chain identifies one or more stimuli and one or more responses, and serves as a definition of relations between groups of events. Time chains can refer to other time chains which are then called time chain segments. The segments are neither parallel nor sequential to their composing timing chain, but act as a decomposition.

A constraint may be associated with the segment to define temporal restrictions on the segments, in terms of maximal delays or minimal reactions. Constraints may also be defined directly on a set of events, for example setting bounds on synchronization or rate.

Figure 13 contains a part of the TADL metamodel. It is based, just like AUTOSAR and EAST-ADL2, on the AUTOSAR metamodeling rules, so an integration in the EAST-ADL2 metamodel is uncomplicated.

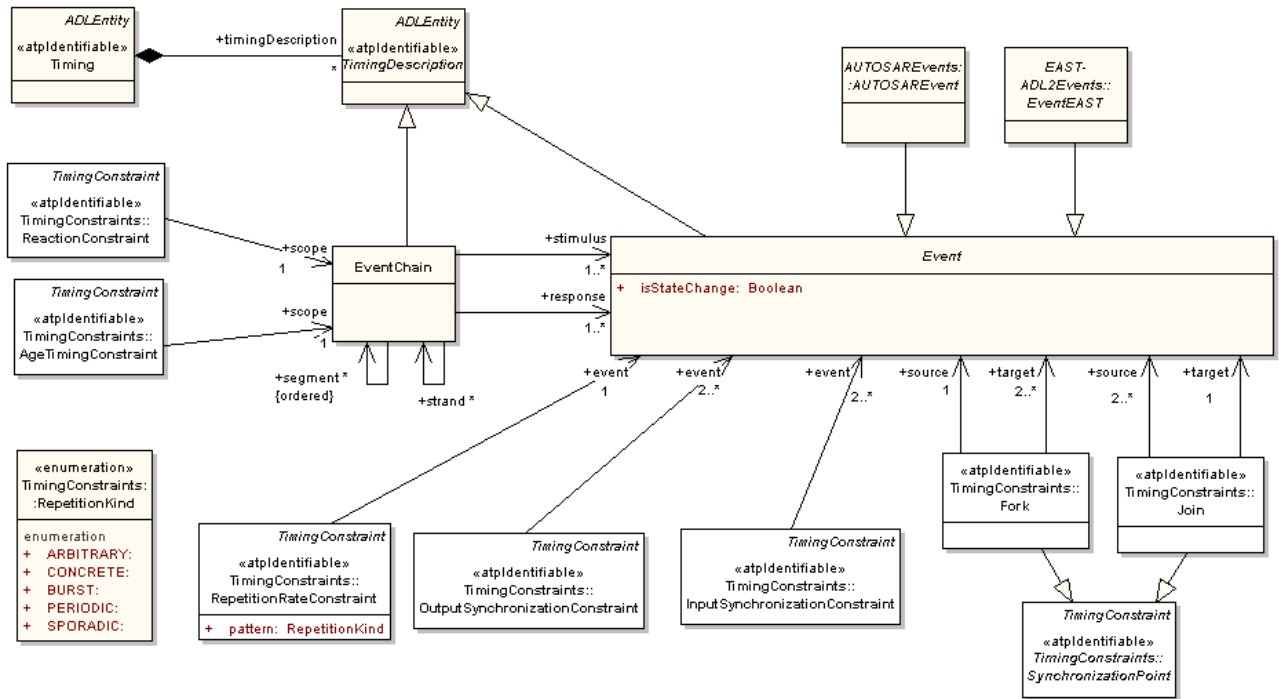


Figure 13. TIMMO timing constructs (excerpt)

3 Contribution to overall ATESST2 objectives

The purpose of this report is to analyse and evaluate the current means to express requirements in the EAST-ADL as defined in the project ATESST. This also includes evaluating the current means of related approaches such as SysML.

4 Conclusions

Requirements are critical elements in a system model, in particular when safety is addressed, as in the ATESSST2 project. For that reason the evaluation of existing requirement approaches is important. Based on what was found in ATESSST, there are several areas that need further refinement in EAST ADL2. New SysML concepts that evolved needs to be included and extended. Timing constructs need to be amended to cover more specification and analysis scenarios. The RIF standard has emerged and should be covered. Means for integration with dedicated requirements engineering tools should be specified. Overall, the basic mechanisms are there, and an ADL is the ideal framework for requirements engineering and the expressiveness it needs.

-

5 Annex 1: SysML

The SysML specification, OMG SysML™ Version 1.0 151 [4], defines a general-purpose modeling language for systems engineering applications, called the OMG Systems Modeling Language (OMG SysML™). Throughout the rest of this document, the language will be referred to as **SysML**.

SysML supports the specification, analysis, design, and verification and validation of a broad range of complex systems.

These systems may include hardware, software, information, processes, personnel, and facilities.

The origins of the SysML initiative can be traced to a strategic decision by the International Council on Systems Engineering's (INCOSE) [5] Model Driven Systems Design workgroup in January 2001 to customize the Unified Modeling Language (UML) for systems engineering applications. This resulted in a collaborative effort between INCOSE and the Object Management Group (OMG) [4], which maintains the UML specification, to jointly charter the OMG Systems Engineering Domain Special Interest Group (SE DSIG)

The SE DSIG, with support from INCOSE and the ISO AP 233 workgroup, developed the requirements for the modeling language, which were subsequently issued by the OMG as part of the UML for Systems Engineering Request for Proposal (UML for SE RFP; OMG document ad/2003-03-41) in March 2003.

Currently it is common practice for systems engineers to use a wide range of modeling languages, tools and techniques on large systems projects. In a manner similar to how UML unified the modeling languages used in the software industry, SysML is intended to unify the diverse modeling languages currently used by systems engineers.

SysML reuses a subset of UML2 and provides additional extensions needed to address the requirements in the UML for SE RFP. SysML uses the UML2 extension mechanisms as the primary mechanism to specify the extensions to UML2.

Since SysML uses UML 2 as its foundation, systems engineers modeling with SysML and software engineers modeling with UML2 will be able to collaborate on models of software-intensive systems. This will improve communication among the various stakeholders who participate in the systems development process and promote interoperability among modeling tools. It is anticipated that SysML will be customized to model domain-specific applications, such as automotive, aerospace, communications, and information systems.

5.1.1 Normative References

[1], [2] and [3] are normative documents that contain provisions which constitute provisions of the SysML specification. Refer to the OMG site [4] for subsequent amendments to, or revisions of any of these publications.

5.1.2 Relationships to Other standards

SysML is defined as an extension of the OMG UML2 Superstructure specification [1].

SysML is intended to be supported by two evolving interoperability standards including the OMG XMI 2.1 model interchange standard for UML2 modeling tools and the ISO 10303 STEP AP233 data interchange standard for systems engineering tools.

SysML supports the OMG's Model Driven Architecture (MDA) initiative by its reuse of the UML and related standards.

5.1.3 SysML Language Architecture

SysML reuses a subset of UML2 and provides additional extensions needed to address requirements in the UML for Systems Engineering RFP. The SysML specification documents the language architecture in terms of the parts of UML2 that are reused and the extensions to UML2.

The SysML language reuses and extends many of the packages from UML. The set of UML meta-classes to be reused are merged into a single meta-model package, UML4SysML.

Some UML packages are not being reused, since they are not considered essential for systems engineering applications.

The SysML profile specifies the extensions to UML. It references the UML4SysML package, thus importing all the meta-classes into SysML that are either reused as-is from UML or extended in SysML. The semantics of UML profiles ensure that when a user model “strictly” applies the SysML profile, only the UML meta-classes referenced by SysML are available to the user of that model. If the profile is not “strictly” applied, then additional UML meta-classes which were not explicitly referenced may also be available. The SysML profile also imports the Standard Profile L1 from UML to make use of its stereotypes.

SysML stereotypes define new modeling constructs by extending existing UML 2 constructs with new properties and constraints. SysML diagram extensions define new diagram notations that supplement diagram notations reused from UML2. SysML model libraries describe specialized model elements that are available for reuse.

The SysML user model is created by instantiating the meta-classes and applying the stereotypes specified in the SysML profile and sub-classing the model elements in the SysML model library.

5.1.4 SysML Language Formalism

The SysML specification is defined by using UML2 specification techniques. These techniques are used to achieve the following goals in the specification.

- Correctness
- Precision
- Conciseness
- Consistency
- Understandability

The specification technique used in the SysML specification [4] describes SysML as a UML extension that is defined using stereotypes and model libraries.

5.1.5 Levels of Formalism

SysML is specified using a combination of UML modeling techniques and precise natural language to balance rigor and understandability. Use of more formal constraints and semantics may be applied in future versions of SysML to further increase the precision of the language.

5.1.6 State of SysML Requirement and Test Case Concepts

The crosscutting constructs that applies to requirements in the SysML specification [4], specifies constructs for system requirements and their relationships

A requirement specifies a capability or condition that must (or should) be satisfied. A requirement may specify a function that a system must perform or a performance condition a system must

achieve. SysML provides modeling constructs to represent text-based requirements and relate them to other modeling elements. The requirements diagram can depict the requirements in graphical, tabular, or tree structure format. A requirement can also appear on other diagrams to show its relationship to other modeling elements. The requirements modeling constructs are intended to provide a bridge between traditional requirements management tools and the other SysML models.

A requirement is defined as a stereotype of UML Class subject to a set of constraints. A standard SysML requirement includes tag value specification fields to specify its unique identifier and text requirement. Additional tag value specification fields such as verification status can be specified by the user.

Several requirements relationships are specified that enable the modeler to relate requirements to other requirements as well as to other model elements. These include relationships for defining a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, and refining requirements.

A composite requirement can contain sub-requirements in terms of a requirements hierarchy, specified using the UML namespace containment mechanism. This relationship enables a complex requirement to be decomposed into its containing child requirements. A composite requirement may state that the system shall do A and B and C, which can be decomposed into the child requirements that the system shall do A, the system shall do B, and the system shall do C. An entire specification can be decomposed into children requirements, which can be further decomposed into their children to define the requirements hierarchy.

Since the concept of requirements reuse is very important in many applications, SysML introduces the concept of a slave requirement. A slave requirement is a requirement whose text property is a read-only copy of the text property of a master requirement. The text property of the slave requirement is constrained to be the same as the text property of the related master requirement. The master/slave relationship is indicated by the use of the copy relationship.

The “derive requirement” relationship relates a derived requirement to its source requirement. This typically involves analysis to determine the multiple derived requirements that support a source requirement. The derived requirements generally correspond to requirements at the next level of the system hierarchy.

The satisfy relationship describes how a design or implementation model satisfies one or more requirements. A system modeler specifies the system design elements that are intended to satisfy the requirement.

The verify relationship defines how a test case verifies a requirement. In SysML, a test case is intended to be used as a general mechanism to represent any of the standard verification methods for inspection, analysis, demonstration, or test.

Additional subclasses can be defined by the user if required to represent the different verification methods. A verdict property of a test case can be used to represent the verification result. The SysML test case is defined consistent with the UML testing profile to facilitate integration between the two profiles.

The refine requirement relationship can be used to describe how a model element or set of elements can be used to further refine a requirement. For example, a use case or activity diagram may be used to refine a text-based functional requirement. Alternatively, it may be used to show how a text-based requirement refines a model element. In this case, some elaborated text could be used to refine a less fine-grained model element.

A generic trace requirement relationship provides a general-purpose relationship between a requirement and any other model element. The semantics of trace include no real constraints and therefore are quite weak. As a result, it is recommended that the trace relationship not be used in conjunction with the other requirements relationships described above.

The rationale construct is quite useful in support of requirements. It enables the modeler to attach a rationale to any requirements relationship or to the requirement itself.

Modelers can customize requirements taxonomies by defining additional subclasses of the Requirement stereotype. The stereotype enables the modeler to add constraints that restrict the types of model elements that may be assigned to satisfy the requirement.

The Requirement Diagram can only display requirements, packages, other classifiers, test cases, and rationale. The relationships for containment, deriveReq, satisfy, verify, refine, copy, and trace can be shown on a requirement diagram.

The callout notation can also be used to reflect the relationship of other model elements to a requirement.

The «requirement» compartment label for the stereotype properties compartment (e.g., id and text) can be elided.

A callout notation can be used to represent derive, satisfy, verify, refine, copy, and trace relationships. For brevity, the «elementType» may be elided.

Requirements can also be represented on other diagrams to show their relationship to other model elements. The compartment and callout notation described above can be used. The callouts represent the requirement that is attached to another model element such as a design element.

The requirements table: the tabular format is used to represent the requirements, their properties and relationships, and may include:

- Requirements with their properties in columns.
- A column that includes the supplier for any of the dependency relationships (Derive, Verify, Refine, Trace).
- A column that includes the model elements that satisfy the requirement.
- A column that represents the rationale for any of the above relationships, including reference to analysis reports for trace rationale, trade studies for design rationale, or test procedures for verification rationale.

The relationships between requirements and other objects can also be shown using a sparse matrix style. The table should include the source and target elements names (and optionally kinds) and the requirement dependency kind.

5.1.6.1 SysML – Requirement Construct, «requirement»

The requirement construct in SysML [4] specifies a capability or condition that must (or should) be satisfied. The requirement construct may specify a function that a system must perform or a performance condition that a system must satisfy.

The requirements constructs are used to establish a contract between the customer (or other stakeholder) and those responsible for designing and implementing the system.

A requirement in SysML is a stereotype of UML class [4]. Compound SysML requirements can be created by using the nesting capability of the class definition mechanism. The default interpretation of a compound requirement, unless stated differently by the compound requirement itself, is that all its sub-requirements must be satisfied for the compound requirement to be satisfied.

Sub-requirements can be accessed through the “nestedClassifier” property of a UML class. When a requirement has nested requirements, all the nested requirements apply as part of the container requirement. Deleting the container requirement deleted the nested requirements, a functionality inherited from UML.

The following tag value specifications apply to the SysML requirement construct:

- [1] text: String; The textual representation or a reference to the textual representation of the requirement.
- [2] id: String; The unique id of the requirement.
- [3] /satisfiedBy: NamedElement [*]; Derived from all elements that are the client of a SysML satisfy requirement relationship for which this requirement is a supplier.
- [4] /verifiedBy: NamedElement [*]; Derived from all elements that are the client of a SysML verify requirement relationship for which this requirement is a supplier.
- [5] /tracedTo: NamedElement [*]; Derived from all elements that are the client of a SysML trace requirement relationship for which this requirement is a supplier.
- [6] /derived: Requirement [0..1]; Derived from all requirements that are the client of a SysML derive requirement relationship for which this requirement is a supplier.
- [7] /derivedFrom: Requirement [*]; Derived from all SysML requirements that are the supplier of a SysML derive requirement relationship for which this requirement is a client.
- [8] /refinedBy: NamedElement [*]; Derived from all elements that are the client of a SysML refine requirement relationship for which this requirement is a supplier.
- [9] /master: Requirement [0..1]; This is a derived property that lists the master SysML requirement for this slave requirement. The master attribute is derived from the supplier of the SysML copy dependency that has this requirement as the slave.

The following constraints apply to the SysML requirement construct:

- [1] The property “isAbstract” must be set to true.
- [2] The property “ownedOperation” must be empty.
- [3] The property “ownedAttribute” must be empty.
- [4] Classes stereotyped by SysML «requirement» may not participate in associations.
- [5] Classes stereotyped by SysML «requirement» may not participate in generalizations.
- [6] A nested classifier of a class stereotyped by SysML «requirement» must also be stereotyped by «requirement».

5.1.6.1.1 *Disposition: Resolved OMG Issue No: 11490 Title: Requirements are abstract*

Source:

No Magic, Inc. (Mr. Nerijus Jankevicius, nerijus@nomagic.com)

Summary:

The requirement construct in SysML is abstract (isAbstract must be true).

However name of the SysML requirement stereotypes are not displayed in italic as defined in UML notation.

Resolution:

(This discussion is elaborated from RTF Telecon Minutes 2008-02-13 & 2008-02-13)

This issue asks that the name of a SysML requirement stereotype to be in italics, in keeping with the rules on UML classes when isAbstract is True. This raises the following questions:

1. Do SysML requirement construct need to be abstract?
2. Do SysML requirement constructs need subclasses? SysML does not currently support any form of sub-classing of requirements.
3. How robustly do SysML requirement constructs need to support properties? Are static properties necessary?

One philosophy considers that if SysML requirements constructs had any features such as properties, they could be only static features that applied to the entire requirement class and not to any instance, which was part of the rationale for the `isAbstract = True` constraint.

If static properties were supported on SysML requirements, along with adding specialization (sub-classing) relationships between requirements, they could support a more complete form of property-based requirements in addition to their current support for text-based requirements. This would more closely parallel the requirements capability in STEP AP233, which has support for both text- and property-based requirements. It would also allow performance requirements to be stated by defined property values that could participate in parametrics or other analysis.

In the original submission, however, the SysML model of requirements has been left deliberately simple without further detail for modeling the system itself, in part so SysML would more closely match the scope and structure of typical requirements specifications which have simple containment models of text-based statements. Associating properties with requirements relied on linking requirements to highly abstract models of system structure, built using SysML blocks, and providing a skeleton model of system structure to which properties would belong. This is appropriate because the properties are typically of the system, rather than of the requirement. This approach also provides a more extensive means to capture properties and other details of an evolving system very early in its specification process. This option remains available in SysML to model greater detail about requirements that need to go to a greater level of detail or precision.

This remains a clear and consistent approach to SysML requirements, and should not be reconsidered at this time. The decision of this resolution is not to add any additional support for requirements sub-classing or properties in this revision of SysML. Separate issues could be raised for future revisions of SysML to consider adding requirements sub-classing or properties, but such an addition is outside the scope of this issue, which seeks only to make the font of requirement names consistent with the `isAbstract` attribute.

Given the lack of sub-classing for SysML requirements, the constraint on `isAbstract` has no added semantics and could be constrained either way. The main issue at this point is simply whether the names of SysML requirements should be in italics or not. Forcing the names into italics, to be consistent with the current constraint, would change the notation defined by the current specification and used in all the current requirements examples. Removing the current constraint will allow tools to keep the name consistent with the setting of the `isAbstract` attribute. The same resolution should apply to Viewpoint in Chapter 7 in the SysML specification [4], which has the same constraint on the `isAbstract` attribute.

Revised Text:

Section 7.3.2.5, under Constraints in the SysML specification [4], remove:
“[4] The property `isAbstract` must be set to true.”

Section 16.3.2.3, under Constraints in the SysML specification [4], remove:
“[1] The property “`isAbstract`” must be set to true.”
And renumber following constraints.

The requirement related construct in SysML [4] is used to add properties to those elements that are related to SysML requirements via the various SysML dependencies. The tagged value specifications are shown using callout notation.

The following tag value specifications apply to the SysML requirement related construct:

- [1] /satisfies: Requirement [*]; Derived from all SysML requirements that are the supplier of a SysML satisfy requirement relationship for which this element is a client.
- [2] /refines: Requirement [*]; Derived from all SysML requirements that are the supplier of a SysML refine requirement relationship for which this element is a client.
- [3] /tracedFrom: Requirement [*]; Derived from all SysML requirements that are the supplier of a SysML trace requirement relationship for which this element is a client.

5.1.6.3 SysML – Copy Requirement Relationship, «copy»

The SysML copy requirement relationship [4] is a dependency construct between a supplier requirement and a client requirement that specifies that the text of the client requirement is a read-only copy of the text of the supplier requirement.

A copy dependency created between two requirements maintains a master/slave relationship between the two elements for the purpose of requirements re-use in different contexts. When a copy dependency exists between two requirements, the requirement text of the client requirement is a read-only copy of the requirement text of the requirement at the supplier end of the dependency.

The following constraints apply to the SysML copy relationship:

- [1] A SysML copy dependency may only be created between two classes that have the SysML requirement stereotype, or a subtype of the SysML requirement stereotype applied.
- [2] If the supplier SysML requirement has SysML sub-requirements, copies of the sub-requirements are made recursively in the context of the client requirement and the SysML copy dependencies are created between each sub-requirement and the associated copy.
- [3] The text property of the SysML client requirement is constrained to be a read-only copy of the text property of the SysML supplier requirement.
- [4] Constraint [3] is applied recursively to all SysML sub-requirements.

5.1.6.4 SysML – Derive Requirement Relationship, «deriveReq»

The SysML derive requirement relationship [4] is a dependency construct between two SysML requirements in which a client requirement can be derived from the supplier requirement.

As with other UML dependencies, the arrow direction points from the derived (client) SysML requirement to the (supplier) SysML requirement from which it is derived.

The following constraints apply to the SysML derive requirement relationship:

- [1] The supplier must be an element stereotyped by SysML requirement or one of SysML requirement subtypes.
- [2] The client must be an element stereotyped by SysML requirement or one of SysML requirement subtypes.

5.1.6.5 SysML – Satisfy Requirement Relationship, «satisfy»

The SysML satisfy requirement relationship [4] is a dependency construct between a SysML requirement and a model element that fulfils the requirement.

As with other UML dependencies [4], the arrow direction points from the satisfying (client) model element to the (supplier) requirement that is satisfied.

The following constraint applies to the SysML satisfy requirement relationship:

[1] The supplier must be an element stereotyped by SysML requirement or one of SysML requirement subtypes.

5.1.6.5.1 Disposition: Resolved OMG Issue No: 11269 Title: «satisfy» is displayed as dependency

Source:

oose Innovative Informatik GmbH (Mr. Tim Weilkiens, tim.weilkiens@oose.de)

Summary:

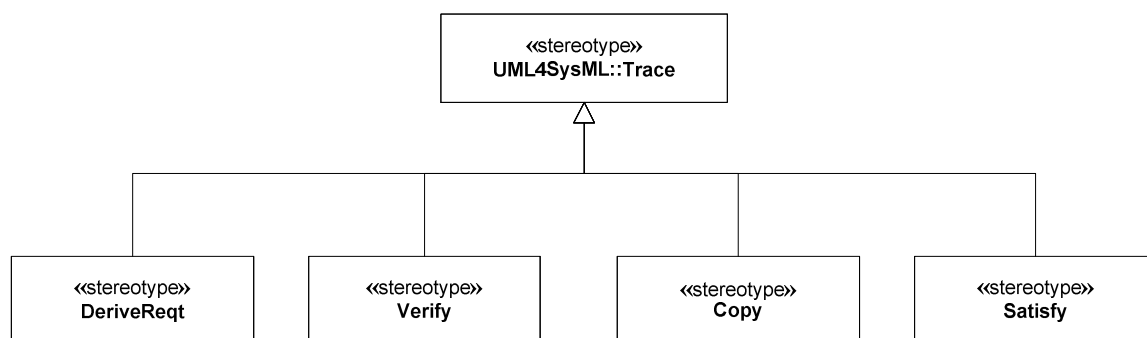
It is not explicitly mentioned that SysML changes the notation for the satisfy relationship. It is a stereotyped realization relationship and should be notated as a dashed line with a triangular arrowhead. But SysML uses a simple arrowhead.

Resolution:

The desired notation is that a «satisfy» dependency be displayed with an open arrowhead like all the other dependencies between SysML requirements. Rather than specialize Realization, which carries its notation of a closed arrowhead, change the abstract syntax shown in Figure 16.1 in the SysML specification [4] to have Satisfy specialize UML Trace, like all the other dependencies defined in this figure.

Revised Text:

Change the upper part of Figure 16.1 in the SysML specification [4] to the following:



5.1.6.6 SysML – Verify Requirement Relationship, «verify»

The SysML verify requirement relationship [4] is a dependency between a requirement and a SysML test case construct that can determine whether a system fulfils the requirement.

As with other UML dependencies [4], the arrow direction points from the (client) test case to the (supplier) requirement.

The following constraints apply to the SysML verify requirement relationship:

[1] The supplier must be an element stereotyped by SysML requirement or one of SysML requirement subtypes.

- [2] The client must be an element stereotyped by SysML testCase or one of the SysML testCase subtypes.

5.1.6.6.1 *Disposition: Resolved OMG Issue No: 11652 Title: Relax constraints on verify relationship*

Source:

Lockheed Martin (Sanford Friedenthal, sanford.friedenthal@lmco.com)

Summary:

In the SysML spec (formal/2007-09-01), the verify relationship currently constrains one of its ends to be a test case. There have been several cases where this is too restrictive. In particular, sometimes it is useful to verify a requirement via analysis and leverage parametrics. In this case, one may want to verify the requirement using a constraint block, block, or constraint property. The proposal is to delete the constraint that a SysML requirement can only be verified with a SysML test case.

Resolution:

Accept the proposal to delete the constraint.

Revised Text:

The proposal is to delete the constraint that a SysML requirement can only be verified with a SysML test case.

1. In 16.3.2.7, pg 152, in the SysML specification [4] delete the following constraint:

[2] The client must be an element stereotyped by SysML testCase or one of the SysML testCase subtypes.
2. In addition, the description in 16.3.2.7, pg 152 in the SysML specification [4] should be changed as follows:

From: A Verify relationship is a dependency between a requirement and a test case that can determine whether a system fulfills the requirement. As with other dependencies, the arrow direction points from the (client) test case to the (supplier) requirement.

To: A Verify relationship is a dependency between a requirement and a test case or other model element that can determine whether a system fulfills the requirement. As with other dependencies, the arrow direction points from the (client) element to the (supplier) requirement.

Note: There were two changes to the above description text: a) added “or other model element”, and b) replaced “test case” by “element.”
3. In Figure 16.1 on pg 148, in the SysML specification [4], change the stereotype property for Requirement as follows:

From: /VerifiedBy: TestCase[*]
To: /verifiedBy: NamedElement [*]
4. Move the stereotype property “/verifies” from TestCase in Figure 16.1 on page 148 in the SysML specification [4] to RequirementRelated in Figure 16.2 on page 149 in the same SysML specification.

In Table 16.2 in the SysML specification [4]:

 - a) Verify Dependency row

change TestCase to NamedElement.

b) Verify Callout row :

2 occurrences- change <<testcase>> TestCaseName to NamedElement

5. In 16.1 5th paragraph in the SysML specification [4] from the end:

a) Insert “or other named element” after “The verify relationship defines how a test case”

b) Replace “a test case is intended to be used” with a “test case or other named element can be used” in the following sentence:

In SysML, a test case is intended to be used as a general mechanism to represent any of the standard verification methods for inspection, analysis, demonstration, or test.

5.1.6.7 SysML – Test Case Construct, «testCase»

The test case construct in SysML [4] applies to a test case that is a method for verifying a SysML requirement is satisfied.

The following tag value specification applies to the SysML test case construct:

[1] /verifies: Requirement [*]; Derived from all SysML requirements that are the supplier of a SysML verify requirement relationship for which this element is a client.

The following constraints apply to the SysML test case construct:

[1] The type of return parameter of the stereotyped model element must be VerdictKind. (note this is consistent with the UML Testing Profile).

6 Annex 2: EAST-ADL2 Requirements

Requirements do not constitute an independent view in EAST-ADL2. In a general ADL architecture, requirement entities will be parts of already existing artifacts. The present framework will show several constructs, which could be used, in principle, in any artifact (VFM, FAA, FDA..). Specific constraints on this will be shown in the related sections.

A requirement expresses a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed properties.

Requirements can be introduced in different phases of the development process for different reasons. They could be introduced by marketing people, control engineers, system engineers, software engineers, Driver/OS developers, basic software developers or hardware engineers. This leads to the fact that requirements have many sources, and requirements are of many types (at different level of detail) and have several relations between them. Under these conditions the number of requirements can become quickly unmanageable if appropriate management does not exist. Note that, requirements can change during the project development and the changes should be taken into account. Requirements are organized hierarchically through several kinds of refinement relations.

EAST-ADL2 has constructs that deal with these problems. Some of these constructs are constructs that deals with general issues in software development and have been already addressed in the past by general processes, like IBM RUP (IBM Rational Unified Process). Here, RUP will not be intended as a guideline; concepts and definitions from RUP will though be used when useful whereas new ones will be introduced when required.

As done for the structure part of EAST-ADL2, the requirements part will be compliant with UML2. The EAST-ADL2 adapts existing concepts whenever possible and develops new ones otherwise. And important source on the level of requirements is the Requirements Interchange Format (RIF).

The purpose of the metaclasses in the Requirements domain meta-model package is to specify rigorously ("formally") the Requirements concepts for the specific domain.

Support for requirement modeling is provided by the EAST-ADL2 on two levels: a generic level and a specialized level where specialized requirement entities are provided which are intended for certain special uses.

6.1.1 EAST-ADL2 - Requirements Construct, «ADLRequirement»

The requirement construct in EAST-ADL2 can be seen as the automotive specialization of the SysML requirement construct [4]. It specifies a capability or condition that must (or should) be satisfied. The requirement construct may specify an automotive function that an automotive system must perform or a performance condition that this system must satisfy.

The EAST-ADL2 requirements constructs are used to establish a contract between the customer (or other stakeholder) and those responsible for designing and implementing the automotive system.

The requirement construct in EAST-ADL2 is also a specialization of the EAST-ADL2 construct for traceable specifications. This enables the EAST-ADL2 requirement construct to be pointed out as an attribute in a specific EAST-ADL2 context (traceable specification tag).

The following attributes apply to the EAST-ADL2 requirement construct:

- [1] text: String; The textual representation or a reference to the textual representation of the requirement.
- [2] id: String; The unique id of the requirement.
- [3] applicability: String; Defines the conditions under which the requirement should hold.

The following constraint applies to the EAST-ADL2 requirement construct:

- [1] The SysML derived tag value specifications /satisfiedBy, /verifiedBy, /tracedTo, /derived, /derivedFrom, /refinedBy, and /master are constrained to not to be used in the EAST-ADL2 requirement construct. The SysML requirement construct is aware of its target block; this is “disabled” in the EAST-ADL2 requirement construct.

6.1.2 EAST-ADL2 – Requirements Container Construct, «ADLRequirementContainer»

The requirement container construct in EAST-ADL2 is used to aggregate several EAST-ADL2 requirements which are semantically related to each other. It can be seen as a larger unit or module of specification information.

The requirement container construct in EAST-ADL2 is a specialization of the EAST-ADL2 construct for traceable specifications. This enables the requirement container to be pointed out as an attribute of a specific EAST-ADL2 context (traceable specification tag).

The following attribute applies to the EAST-ADL2 requirement container construct:

- [1] childRequirements : ADLRequirement [0..*] {ordered}; The child requirements of this EAST-ADL2 requirement container.

6.2 Annex 3: EAST-ADL2 Infrastructure

This part describes the structure and content of the Infrastructure package of EAST-ADL2. It defines general-purpose constructs and types that may be used to model structural constructs in several modeling diagrams.

The purpose of the metaclasses in the Infrastructure domain metamodel package is to specify rigorously (“formally”) the core concepts for the specific domain. Each following section contains a specification for each individual metaclass owned in the Infrastructure package.

6.2.1 Package Context

The Context subpackage of the Infrastructure::CoreConstruct package of EAST-ADL2 extends the basic element to support ownership and associations of other elements.

6.2.1.1 EAST-ADL2 – Construct for Context , «ADLContext»

The EAST-ADL2 context construct is used to achieve a simple and practical way to allocate traceable specifications to a specific EAST-ADL2 model context and to let this specific model context own its comments and relationships.

The following attributes apply to the EAST-ADL2 context construct:

- [1] ownedRelationship : ADLRelationship [0..*]; Relationship(s) owned in this context.
- [2] traceableSpecification : ADLTraceableSpecification [0..*]; Traceable specification(s) allocated to this context.
- [3] ownedComment: ADLComment[0..*]; Comment(s) owned in this context.

6.2.2 Package Elements

The Element subpackage of the Infrastructure:CoreConstruct package of the EAST-ADL2 specifies the most basic abstract constructs, ADLEntity and ADLTraceableSpecification.

6.2.2.1 EAST-ADL2 – Construct for Entity, «ADLEntity«

ADLEntity is an abstract metaclass that represents an arbitrary named entity in the domain model. The name is used for identification of the element within the namespace in which it is defined.

The name attribute is used for identification of the ADL entity within the namespace where its name is accessible. Note that the attribute has a multiplicity of [0..1] which provides for the possibility of the absence of a name (which is different from the empty name).

Also the ADLEntity can be used to extend the EAST-ADL approach to other languages and standards by adding a generalize relation from the respective (non EAST-ADL) element to the ADLEntity, by that be able to participate in e.g. the ADLSatisfy and ADLRealization dependencies (Extension Point).

6.2.2.2 EAST-ADL2 – Construct for Traceable Specifications, «ADLTraceableSpecification«

In many cases, EAST-ADL entities such as an ADLFunction or a Sensor need to contain requirements and other specification elements as a part of the entity. In other cases, the relation between the entity and the related specification element is specific for a certain context: for example a requirement on a sensor is only applicable in a certain hardware architecture. For this reason, the traceablespecification concept with traceablespecification, ADLRelationship and ADLContext was introduced. The EAST-ADL2 construct for traceable specifications is the superclass for the specification entities of the language. A specialization of a traceable specification is for example the EAST-ADL2 requirement construct.

6.2.3 Package Relationships

The Infrastructure::CoreConstructs::Relationships subpackage of EAST-ADL2 defines general-purpose relationship constructs that may be used to model dependencies between structural constructs.

The purpose of the metaclasses in the Infrastructure::CoreConstructs::Relationships domain metamodel package is to specify rigorously ("formally") the various relationships that may exist between ADL basic constructs.

6.2.3.1 EAST-ADL2 – Construct for Relationships, «ADLRelationship«

The EAST-ADL2 construct for relationships makes it possible for various relationships in the language to be owned by a specific context. A specialization of a relationship is for example the EAST-ADL2 satisfy requirement relationship.

6.2.3.2 EAST-ADL2 – Derive Requirements Relationship, «ADLDeriveReq«

The derive requirement relationship in EAST-ADL2 is an automotive specialization of the SysML derive requirement relationship [4]. The EAST-ADL2 derive requirement relationship signifies a dependency relationship in-between two sets of EAST-ADL2|SysML requirement constructs, showing the relationship when a set of derived client EAST-ADL2|SysML requirements are derived from a set of supplier EAST-ADL2|SysML requirements.

As with other UML dependencies [4] the dependency arrow direction points from the derived (client) EAST-ADL2|SysML requirement, to the (supplier) EAST-ADL2|SysML requirement.

The following constraints apply to the EAST-ADL2 derive requirement relationship:

- [1] The first and second constraint in the SysML specification related to derive requirement relationship [4] is not applicable.
- [2] The supplier must be an element stereotyped by EAST-ADL2|SysML requirement or one of EAST-ADL2|SysML requirement subtypes.
- [3] The client must be an element stereotyped by EAST-ADL2|SysML requirement or one of EAST-ADL2|SysML requirement subtypes.

6.2.3.3 EAST-ADL2 – Refine Requirements Relationship, «ADLRefine»

The refine requirement relationship in EAST-ADL2 signifies a dependency relationship in-between an EAST-ADL2|SysML requirement and an EAST-ADL2 entity, showing the relationship when a client EAST-ADL2 entity construct refines the supplier EAST-ADL2|SysML requirement.

As with other UML dependencies [4] the arrow direction points from the EAST-ADL2 entity that refines the EAST-ADL2|SysML requirement to the refined (supplier) EAST-ADL2|SysML requirement.

The following constraints apply to the EAST-ADL2 refine requirement relationship:

- [1] The supplier must be an element stereotyped by EAST-ADL2|SysML requirement or one of EAST-ADL2|SysML requirement subtypes.
- [2] The client must be an element stereotyped by EAST-ADL2 entity construct or one of EAST-ADL2 entity subtypes.

6.2.3.4 EAST-ADL2 – Satisfy Requirements Relationship, «ADLSatisfy»

The satisfy requirement relationship in EAST-ADL2 is an automotive specialization of the SysML satisfy requirement relationship [4]. The satisfy requirement relationship in EAST-ADL2 signifies a dependency relationship in-between an EAST-ADL2|SysML requirement and a satisfying EAST-ADL2|AUTOSAR entity, showing the relationship when a client EAST-ADL2|AUTOSAR entity construct satisfies the supplier EAST-ADL2|SysML requirement.

As with other UML dependencies the dependency arrow direction points from the (client) EAST-ADL2|AUTOSAR entity to the satisfied (supplier) EAST-ADL2|SysML requirement.

The following constraints apply to the EAST-ADL2 satisfy requirement relationship:

- [1] The first constraint in the SysML specification related to the satisfy requirement relationship [4] is not applicable.
- [2] The supplier must be an element stereotyped by EAST-ADL2|SysML requirement or one of EAST-ADL2|SysML requirement subtypes.
- [3] The client must be an element stereotyped by EAST-ADL2|AUTOSAR entity construct.
- [4] The client may not be an EAST-ADL2 requirement construct or an EAST-ADL2 requirement container construct.

6.2.3.5 EAST-ADL2 – Verify/Validate Requirements Relationship, «ADLVerify»

The verify/validate requirement relationship in EAST-ADL2 is an automotive specialization of the SysML verify requirement relationship [4]. The verify/validate requirement relationship in EAST-ADL2 signifies a dependency relationship in-between an EAST-ADL2|SysML requirement and a

verifying/validating EAST-ADL2|AUTOSAR entity, showing the relationship when a client EAST-ADL2|AUTOSAR entity constructs verify/validate the supplier EAST-ADL2|SysML requirements.

As with other UML dependencies [4] the dependency arrow direction points from the (client) EAST-ADL2|AUTOSAR entity to the (supplier) EAST-ADL2|SysML requirement.

The following constraints apply to the EAST-ADL2 verify/validate requirement relationship:

- [1] The first and second constraint in the SysML specification related to the verify requirement relationship [4] is not applicable.
 - [2] The supplier must be an element stereotyped by EAST-ADL2|SysML requirement or one of EAST-ADL2|SysML requirement subtypes.
 - [3] The client must be an element stereotyped by EAST-ADL2|AUTOSAR entity construct.
 - [4] The client may not be an EAST-ADL2 requirement construct or an EAST-ADL2 requirement container construct.
-
-

7 References

- [1] Unified Modeling Language: Infrastructure, version 2.1.1 (<http://www.omg.org/cgi-bin/doc?formal/2007-02-06>)
- [2] MOF 2.0/XMI Mapping Specification, v2.1 (<http://www.omg.org/cgi-bin/doc?formal/2005-09-01>)
- [3] OMG: www.omg.org
- [4] INCOSE: www.incose.org
- [5] EAST-EEA: <http://www.east-eea.net/>
- [6] ATESST: <http://www.atesst.org>
- [7] AUTOSAR: <http://www.autosar.org>
- [8] www.sysml.org
- [9] EAST-ADL version 1.0
- [10] EAST-ADL version 2.0
- [11] AUTOSAR Specification of the Virtual Functional Bus, Version 1.0.1, Rev. 0001 (part of AUTOSAR Release 3.1)
- [12] AUTOSAR Timing Meta Model, Version 0.95, 2005-06-28 (internal draft document)
- [13] AUTOSAR Timing Model SPS (Software Problem Summary), Version 0.1, 2005-06-10 (internal draft document)
- [14] TIMMO Deliverable D2: "TADL: Timing Augmented Description Language V1"
- [15] MARTE: UML Profile for Modeling and Analysis of Real-Time and Embedded systems. <http://www.omgarte.org/>
- [16] www.timmo.org